

HENRIQUE PIRES DE ALMEIDA NOBRE

GERAÇÃO AUTOMÁTICA DE MÓDULOS VHDL PARA LOCALIZAÇÃO  
DE PADRÕES INVARIANTE A ESCALA E ROTAÇÃO EM FPGA

Dissertação apresentada à Escola  
Politécnica da Universidade de São Paulo  
para obtenção do Título de Mestre em  
Engenharia Elétrica.

São Paulo  
2009

HENRIQUE PIRES DE ALMEIDA NOBRE

GERAÇÃO AUTOMÁTICA DE MÓDULOS VHDL PARA LOCALIZAÇÃO  
DE PADRÕES INVARIANTE A ESCALA E ROTAÇÃO EM FPGA

Dissertação apresentada à Escola  
Politécnica da Universidade de São Paulo  
para obtenção do Título de Mestre em  
Engenharia Elétrica.

Área de Concentração:  
Microeletrônica

Orientador:  
Prof. Dr. Hae Yong Kim

São Paulo  
2009

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 20 de abril de 2009

Assinatura do autor

Assinatura do orientador

## FICHA CATALOGRÁFICA

**Nobre, Henrique Pires de Almeida**

**Geração Automática de Módulos VHDL para Localização de Padrões Invariante d Escala d Rotação em FPGA**

**/ H.P. de A. Nobre. -- São Paulo, 2009.**

**167 p.**

**Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.**

**1.Processamento de Imagem 2.FPGAs 3.Template Matching  
4.VHDL I.Universidade de São Paulo. Escola Politécnica.  
Departamento de Engenharia de Sistemas Eletrônicos II.t.**

Todo o trabalho, esforços e sacrifícios que amalgamados, conceberam esta dissertação, são, todos, dedicados à minha esposa, mãe e pai; eternos e incondicionais compartilhadores das trilhas e resultados dos desafios aceitos por mim. Eventuais incorreções são porém, de minha inteira responsabilidade.

## **AGRADECIMENTOS**

Permaneci pouco mais de dois terços de meu mestrado ligado ao Grupo de Eletrônica Molecular (GEM) da Politécnica. Agradeço ao grupo pelo companheirismo, em especial, ao meu orientador deste período, o Prof. Dr. Fernando Fonseca pela ajuda, amizade e compreensão.

Agradeço ao meu orientador na fração final do meu mestrado, o Prof. Dr. Hae Yong Kim, pelo voto de confiança em meu comprometimento para a conclusão de um novo projeto de mestrado em quase um ano. O apoio destes dois pesquisadores tornou este trabalho possível.

Agradeço à minha esposa Grazielle, meu porto seguro, pela compreensão, pelo sacrifício, dedicação, conselhos e pelo amor imortal em todos os momentos.

Aos meus pais agradeço também pelo encorajamento eterno.

Um voto de obrigado deve ser deixado à empresa Optovac pelo tempo cedido e pela infra-estrutura tecnológica oferecida desde o início desta empreitada.

Agradeço ao meu irmão Gustavo pela ajuda, sobretudo referente às pesquisas e livros que tanto me ajudaram nas disciplinas. Obrigado Brodinho.

Meus sinceros agradecimentos a todos os amigos e à família pela compreensão nos duros momentos de ausência, sejam durante férias ou feriados, os quais permitiram agregar maior qualidade a este trabalho. Neste sentido, agradeço principalmente nos momentos finais deste trabalho à minha parceira de cinema, Ana Carolina pela paciência e alegrias e ao amigo Rogério pela ajuda logística.

Agradeço aos amigos Neimar, a quem devo minha introdução mais profissional ao mundo dos FPGAS e pelas sempre valiosas discussões técnicas.

Agradeço ao amigo Dimas pela ajuda com as pesquisas e discussões sobre a linguagem C e os algoritmos.

Este trabalho não teria chance de ser concluído também se não fossem os apoios sobre-humanos dos meus orientadores e ajuda dos amigos da secretaria, Cláudia, Simone e Marcelo, pelos inúmeros salvamentos burocráticos.

Ao Prof. Piqueira deixo meu agradecimento pela compreensão durante meu processo de qualificação.

Finalmente, agradeço ao amigo Dr. Celso Charuri por comprovar-me a possibilidade real e a importância de lapidarmo-nos como indivíduos: o maior, mais extenso e gratificante dos desafios.

O apoio destes amigos, tornaram transponíveis, os duros obstáculos até aqui percorridos.

## RESUMO

A busca por padrões em imagens é um problema clássico em visão computacional e consiste em detectar a presença de uma dada máscara em uma imagem digital. Tal tarefa pode se tornar consideravelmente mais complexa com a invariância aos aspectos da imagem tais como rotação, escala, translação, brilho e contraste (RSTBC - *rotation, scale, translation, brightness and contrast*). Um algoritmo de busca de máscara foi recentemente proposto. Este algoritmo, chamado de Ciratefi, é invariante aos aspectos RSTBC e mostrou-se bastante robusto. Entretanto, a execução deste algoritmo em um computador convencional requer diversos segundos. Além disso, sua implementação na forma mais geral em *hardware* é difícil pois há muitos parâmetros ajustáveis. Este trabalho propõe o projeto de um software que gera automaticamente módulos compiláveis em *Hardware Description Logic* (VHDL) que implementam o filtro circular do algoritmo Ciratefi em dispositivos *Field Programmable Gate Array* (FPGA). A solução proposta acelera o tempo de processamento de 7s (em um PC de 3GHz) para 1,367ms (em um dispositivo Stratix III da Altera). Esta performance excelente (mais do que o necessário em sistemas em tempo-real) pode levar a sistemas de visão computacional de alta performance e de baixo custo.

Palavras-chave: invariante ao RSTBC, localização de máscaras, tempo real, VHDL, FPGA, visão computacional.

## **ABSTRACT**

Template matching is a classical problem in computer vision. It consists in detecting the presence of a given template in a digital image. This task becomes considerably more complex with the invariance to rotation, scale, translation, brightness and contrast (RSTBC). A novel RSTBC-invariant robust template matching algorithm named Ciratefi was recently proposed. However, its execution in a conventional computer takes several seconds. Moreover, the implementation of its general version in hardware is difficult, because there are many adjustable parameters. This work proposes a software that automatically generates compilable Hardware Description Logic (VHDL) modules that implement the circular filter of the Ciratefi template matching algorithm in Field Programmable Gate Array (FPGA) devices. The proposed solution accelerates the time to process a frame from 7s (in a 3GHz PC) to 1.367ms (in Altera Stratix III device). This excellent performance (more than the required for a real-time system) may lead to cost-effective high-performance co-processing computer vision systems.

Keywords: RSTBC-invariant, template matching, real time, VHDL, FPGA, computer vision.

# SUMÁRIO

LISTA DE FIGURAS .....	10
LISTA DE TABELAS.....	13
<b>CAPÍTULO 1. INTRODUÇÃO.....</b>	<b>17</b>
1.1. <b>INTRODUÇÃO.....</b>	<b>17</b>
1.2. <b>MOTIVAÇÃO.....</b>	<b>20</b>
1.3. <b>OBJETIVO.....</b>	<b>23</b>
1.4. <b>ORGANIZAÇÃO DO TRABALHO.....</b>	<b>25</b>
<b>CAPÍTULO 2. REVISÃO DA LITERATURA.....</b>	<b>27</b>
2.1. <b>O ALGORITMO CIRATEFI.....</b>	<b>29</b>
2.1.1. O FILTRO CIFI.....	29
2.1.2. O FILTRO RAFI.....	31
2.1.3. O FILTRO TEFI.....	32
2.1.4. PERFORMANCE DO ALGORITMO CIRATEFI.....	32
2.2. <b>O ALGORITMO SIFT.....</b>	<b>34</b>
<b>CAPÍTULO 3. TECNOLOGIA DE PROCESSAMENTO.....</b>	<b>35</b>
3.1. <b>DISPOSITIVOS DE PROCESSAMENTO PARALELO.....</b>	<b>35</b>
3.1.1. DISPOSITIVOS TIPO ASIC.....	36
3.1.2. MEMÓRIAS PROGRAMÁVEIS DE LEITURA.....	37
3.1.3. DISPOSITIVOS PLA.....	37
3.1.4. DISPOSITIVOS PAL.....	38
3.1.5. DISPOSITIVOS CPLD E FPGA.....	39
3.2. <b>PRINCIPAIS DISPOSITIVOS DE ACELERAÇÃO DE PROCESSAMENTO.....</b>	<b>44</b>
3.2.1. DSPs.....	44
3.2.2. GPUs.....	46
3.2.3. FPGA.....	47
3.3. <b>PROCESSAMENTOS CONVENCIONAIS E EM PARALELO.....</b>	<b>52</b>
3.4. <b>FPGAs COMO CO-PROCESSADORES.....</b>	<b>54</b>
3.5. <b>PROGRAMAÇÃO PARA HARDWARES DE ALTO DESEMPENHO.....</b>	<b>56</b>
3.5.1. PIPELINE.....	56
3.5.2. OS FPGAs "ACHRONIX".....	58
3.6. <b>TECNOLOGIA DO DISPOSITIVO UTILIZADO.....</b>	<b>59</b>
3.6.1. MÓDULOS DE LÓGICA ADAPTATIVA.....	61
3.6.2. BLOCOS DE MATRIZ LÓGICA.....	62
3.6.3. MEMÓRIAS INTERNAS.....	63
3.7. <b>ETAPAS PARA PROJETOS EM FPGA.....</b>	<b>72</b>
3.7.1. ETAPA DE DESCRIÇÃO DE HARDWARE.....	72
3.7.2. ETAPA DE SÍNTESE.....	73
3.7.3. ETAPA DE ROTEAMENTO.....	73
3.7.4. SIMULAÇÕES.....	76
<b>CAPÍTULO 4. METODOLOGIA, MATERIAIS E MÉTODOS.....</b>	<b>79</b>
4.1. <b>OBJETIVOS DA IMPLEMENTAÇÃO EM HARDWARE.....</b>	<b>79</b>
4.2. <b>SOLUÇÃO EM HARDWARE.....</b>	<b>80</b>
4.2.1. SOLUÇÃO PARA O FILTRO CIFI E O FILTRO RAFI.....	81
4.2.2. SOLUÇÃO PARA O ACESSO A MEMÓRIA.....	103
4.3. <b>ESTRATÉGIA PARA A IMPLEMENTAÇÃO.....</b>	<b>107</b>
4.3.1. PROGRAMA EM C COMO GERADOR AUTOMÁTICO DE HARDWARE.....	107
4.4. <b>ESTRATÉGIA PARA AS SIMULAÇÕES.....</b>	<b>112</b>

<b>CAPÍTULO 5. RESULTADOS</b> .....	113
5.1. <b>MÓDULOS SINTETIZADOS</b> .....	115
5.2. <b>CÁLCULOS PARA AS SOMATÓRIAS</b> .....	122
5.3. <b>CÁLCULOS DAS MÉDIAS</b> .....	124
5.4. <b>CÁLCULOS DAS CORRELAÇÕES</b> .....	127
5.5. <b>TESTES COM IMAGENS</b> .....	133
5.5.1. <b>IMAGENS EM BAIXA RESOLUÇÃO</b> .....	133
5.5.2. <b>IMAGENS EM BAIXA RESOLUÇÃO ROTACIONADAS</b> .....	136
5.5.3. <b>IMAGENS EM 640x480</b> .....	138
5.6. <b>RESULTADOS FINAIS</b> .....	141
5.7. <b>DESEMPENHOS</b> .....	142
<b>CAPÍTULO 6. DISCUSSÕES</b> .....	145
6.1. <b>PRECISÃO DOS RESULTADOS</b> .....	145
6.2. <b>MELHORAMENTOS FUTUROS</b> .....	147
6.2.1. <b>TERMOS CONSTANTES NA CORRELAÇÃO</b> .....	147
6.2.2. <b>FPGAs DE ALTO DESEMPENHO</b> .....	149
6.2.3. <b>IDEALIZAÇÃO DO SISTEMA FINAL</b> .....	153
6.2.4. <b>FILTRO CIFI EM TEMPO REAL COM CÂMERA</b> .....	156
6.2.5. <b>CONSIDERAÇÕES PARA IMPLEMENTAÇÃO DO FILTRO RAFI</b> .....	159
<b>CONCLUSÃO</b> .....	160
<b>ARTIGOS GERADOS NO MESTRADO</b> .....	162
<b>REFERÊNCIAS BIBLIOGRÁFICAS</b> .....	163

## LISTA DE FIGURAS

Figura 1. Resultado final do Ciratefi com indicações de escala e rotação. Busca e localização independente do brilho e contraste das imagens envolvidas _____	18
Figura 2. Exemplo da utilização do algoritmo de casamento de imagens SIFT sobre imagens estéreas obtidas pelo veículo <i>Spirit</i> em exploração sobre Marte _____	21
Figura 3. Plataforma de movimentação autômata. Emprega sistema de reconhecimento de objetos com algoritmo SIFT em FPGA _____	22
Figura 4. Testes do sistema em FPGA de reconhecimento de ações humanas _____	22
Figura 5. Área da íris para reconhecimento de padrões com implementação em FPGA _____	22
Figura 6. Exemplos de busca e localização de imagem _____	27
Figura 7. Projeções circulares para diferentes escalas _____	30
Figura 8. Resultado do Filtro Cifi indicando em os pixels candidatos do primeiro grau _____	30
Figura 9. Projeção radial para a escala selecionada _____	31
Figura 10. Resultado do Filtro Rafi _____	31
Figura 11. Resultado do Filtro Tefi indicando os pixels localizados da imagem de máscara, independentemente da escala, rotação, contraste e brilho _____	32
Figura 12. Imagens ilustrando o processamento pelo algoritmo Sift _____	34
Figura 13. Arquitetura de um dispositivo PLA _____	38
Figura 14. Arquitetura de um dispositivo PAL _____	39
Figura 15. Diagrama de uma macrocélula de CPLD MAX3000A da Altera _____	40
Figura 16. Diagrama da arquitetura básica para uma FPGA _____	42
Figura 17. Diagrama de um elemento lógico de uma FPGA Stratix III da Altera _____	43
Figura 18. Diagrama de comparação entre o processamento convencional e processamento paralelo de uma FPGA _____	53
Figura 19. Diagrama para o processamento de dados sem o uso de <i>pipelines</i> _____	57
Figura 20. Diagrama para o processamento de dados com o uso três estágios de <i>pipelines</i> _____	57
Figura 21. Arquitetura das FPGAs Achronix _____	58
Figura 22. Estrutura da arquitetura PicoPIPE desenvolvida pela empresa Achronix _____	58
Figura 23. Vista com os principais recursos das FPGAs da família Stratix III _____	59
Figura 24. Características dos modelos de FPGAs da família Stratix III _____	60
Figura 25. Diagrama em alto nível de uma unidade ALM para a Stratix III _____	61
Figura 26. Diagramas com os modos das entradas para as <i>Lookup Tables</i> _____	61
Figura 27. Estrutura do módulo LAB para a família Stratix III _____	62
Figura 28. Sumário das características para os blocos de memória TriMatrix _____	63
Figura 29. Memória tipo Porta Única _____	63
Figura 30. Memória tipo Porta-Dupla Simples _____	64
Figura 31. Memória tipo Porta Dupla Real _____	64
Figura 32. Implementação em cascata de registradores de deslocamento em blocos de memória _____	64
Figura 33. Modos de operação dos blocos DSP na Stratix III _____	65
Figura 34. Blocos DSP multiplica-soma e acumula _____	66
Figura 35. Diagrama de redes de relógios globais _____	67
Figura 36. Redes de relógio regionais _____	67
Figura 37. Localização dos PLLs na Stratix III _____	68
Figura 38. Diagrama de bloco para os PLL da Stratix III _____	69
Figura 39. Bancos de IO para os dispositivos Stratix III _____	70
Figura 40. Diagrama de um pino de IO _____	71

Figura 41. Fluxo de projeto da Altera	75
Figura 42. Estrutura de topo de Hierarquia para simulação e testes	77
Figura 43. Diagrama de um <i>TestBench</i>	78
Figura 44. Interconexões entre os filtros Cifi e Rafi.	83
Figura 45. Arquitetura para leitura de dados	86
Figura 46. Pixels processáveis para uma imagem A de 640x480 pixels	86
Figura 47. Janela de Processamento para o Filtro Cifi	87
Figura 48. Janela de Processamento para o Filtro Rafi	88
Figura 49. Árvore de Somadores em <i>pipeline</i> para o filtro Cifi	89
Figura 50. Árvore de Somadores em <i>pipeline</i> para o filtro Rafi com o módulo Decodificador	90
Figura 51. Árvore de Somadores em <i>pipeline</i> para o filtro Rafi	92
Figura 52. Cálculos intermediários para a computação da correlação	94
Figura 53. Análise dos sinais digitais envolvidos no cálculo completo da correlação	94
Figura 54. Arquitetura para o cálculo da correlação em paralelo para o Filtro Cifi	96
Figura 55. Arquitetura para o cálculo da correlação em paralelo para o Filtro Rafi	97
Figura 56. Ferramenta <i>MegaWizard</i> para a Raiz Quadrada	99
Figura 57. Imagem da ferramenta <i>MegaWizard</i> com as opções de configuração para a divisão e ao fundo o código VHDL gerado	100
Figura 58. Árvore de seleção de correlações para o Filtro Cifi	101
Figura 59. Árvore de seleção de correlações para o Filtro Rafi	102
Figura 60. Diagrama para o acesso aos dados da memória interna em um ciclo de relógio	104
Figura 61. Exemplo para uma imagem A com resolução de 640x480	105
Figura 62. Imagem com resolução de 640x53 representando o conteúdo das memórias ROM armazenadas dentro da FPGA	106
Figura 63. Fragmento com zoom, respectivo à seleção indicada na figura anterior, da imagem que representa a imagem A armazenada dentro da FPGA	106
Figura 64. Entrada do programa gerador do módulo CWP de cálculo das médias	110
Figura 65. Entrada do programa gerador do módulo de cálculo da correlação	110
Figura 66. Arquivo ".bat" para geração dos códigos VHDL para cada correlação	111
Figura 67. Arquitetura para a infra-estrutura de simulação envolvendo as entradas dos dados da imagem e saída dos resultados	112
Figura 68. Imagem A de verificação completa dos resultados com resolução de 55x53	114
Figura 69. Mesma imagem A de verificação onde os pixels centrais em cores diferentes apontam os pixels processáveis para esta resolução	114
Figura 70. Diagrama RTL sintetizado para os módulos de correlação	115
Figura 71. Diagrama sintetizado para os módulos de correlação com 14 e 12 círculos	116
Figura 72. Diagrama RTL em nível 0 para o Módulo de Seleção da Correlação	117
Figura 73. Diagrama de topo de hierarquia do Filtro Cifi	118
Figura 74. Parte do diagrama sintetizado do módulo de somatória e cálculo das médias	119
Figura 75. Leiaute com o módulo Cifi fisicamente roteado na FPGA selecionada	120
Figura 76. Leiaute com o módulo Cifi completo fisicamente roteado na FPGA utilizando a ferramenta de particionamento	121
Figura 77. Saída gráfica gerada automaticamente para análise de coordenadas para o <i>hardware</i> Janela de Processamento CWP	122
Figura 78. Saída com os pontos para coordenadas que serão transferidos ao programa em VHDL para processamento	123
Figura 79. Resultados dos Cálculos das Médias para a imagem A da figura 69	124

Figura 80. Resultados dos cálculos das médias para a imagem A da figura 69 realizados em <i>hardware</i>	125
Figura 81. Parte do código em VHDL responsável pelo cálculo de divisão	125
Figura 82. Exemplo de uma matriz de processamento CQ gerada pelo <i>software</i> em C gerador de códigos VHDL	127
Figura 83. Arquivo VHDL de topo de hierarquia	128
Figura 84. Resultados dos cálculos das correlações feitas em <i>software</i>	129
Figura 85. Resultados dos cálculos das correlações com cálculo em <i>hardware</i>	129
Figura 86. Resultados dos cálculos das correlações feitas em <i>hardware</i>	130
Figura 87. Resultados das correlações feitas em <i>software</i> que reproduzem cálculo em <i>hardware</i>	130
Figura 88. Resultados dos cálculos das correlações feitas em <i>software</i>	131
Figura 89. Resultados intermediários dos cálculos das correlações	132
Figura 90. Imagem de teste para o simulador com resolução de 160x120	133
Figura 91. Imagem do mapa de pixels candidatos para limiar de 0.90	134
Figura 92. Imagem do mapa de pixels candidatos para limiar de 0.95	134
Figura 93. Resultado informando escala mais apropriada para cada pixel candidato	135
Figura 94. Detalhe do mapa de pixels candidatos com indicação de escalas	135
Figura 95. Imagem A de testes com imagem de máscara rotacionada	136
Figura 96. Imagem do mapa de pixels candidatos para limiar de 0.90	137
Figura 97. Imagem do mapa de pixels candidatos para limiar de 0.95	137
Figura 98. Mapa de pixels com baixo valor de limiar	138
Figura 99. Mapa de pixels com limiar intermediário	139
Figura 100. Imagem do mapa de pixels obtido com um valor de limiar elevado	140
Figura 101. Resultados da melhor compilação e roteamento obtidos	143
Figura 102. Resultados do cálculo da divisão da correlação através da divisão em C	146
Figura 103. Etapas do cálculo das correlações	148
Figura 104. Diagrama de representação dos dispositivos da família Stratix IV	149
Figura 105. Periféricos presentes no Kit de Desenvolvimento da empresa Achronix	152
Figura 106. Kit de desenvolvimento <i>PCI Express Stratix II GX Development Board</i>	155
Figura 107. Kit de desenvolvimento <i>PCI Express</i> com FPGA da Xilinx.	155
Figura 108. Kit de desenvolvimento Stratix III.	156
Figura 109. Placa com entrada de vídeo composto e conector tipo HSMC	157
Figura 110. Diagrama de blocos para a execução do filtro Cifi em tempo real	158

## LISTA DE TABELAS

Tabela 1 Parâmetros utilizados para a validação do sistema .....	142
Tabela 2 Performance e tamanho em unidades lógicas para cada módulo.....	143
Tabela 3 Comparação dos resultados das correlações com precisões diferentes..	145

## LISTA DE ABREVIATURAS

- *ALM* : *Adaptive Logic Module*
- *ALU* : *Arithmetic logic Unit*
- *ASIC* : *Application Specific Integrated Circuit*
- *BLCs* : *Blocos Lógicos Configuráveis*
- *CPU* : *Central Processing Unit*
- *CLB* : *Configurable Logic Block*
- *CPLD* : *Complex Programmable Logic Device*
- *CORDIC* : *COordinate Rotation Digital Computer*
- *CUDA* : *Computer Unified Device Architecture*
- *DSP* : *Digital Signal Processing*
- *DUT* : *Design Under Test*
- *EEPROM* : *Electrically Erasable Programmable Read Only Memory*
- *FFT* : *Fast Fourier Transform*
- *FIFO* : *First-In First-Out*
- *FIR* : *Finite Impulse Response*
- *FPGA* : *Field Programmable Gate Array*
- *GPU* : *Graphics Processing Units*
- *GPP* : *General-Purpose Processor*
- *HSMC* : *High-Speed Mezzanine Connector*
- *HDL* : *Hardware Description Language*
- *IP* : *Intellectual Property*
- *IFFT* : *Inverse Fast Fourier Transform*

- *ISA* : *Instruction Set Architecture*
- *JTAG* : *Joint Test Action Group*
- *LAB* : *Logic Array Block*
- *LE* : *Logic Element*
- *LUT* : *Look-Up Table*
- *LVDS* : *Low Voltage Differential Signaling*
- *MAC* : *Multiply Accumulate*
- *PAL* : *Programmable Array Logic*
- *PLA* : *Programmable Logic Array*
- *PLL* : *Phase-Locked Loops*
- *PROMs* : *Programmable Read Only Memories*
- *RAM* : *Random Access Memory*
- *ROM* : *Read Only Memory*
- *RTL* : *Register Transfer Level*
- *RTSBC* : *Rotation, Scale, Translation, Brightness And Contrast*
- *SAD* : *Sum of Absolute Differences*
- *SIFT* : *Scale Invariant Feature Transform*
- *SRAM* : *Static Random Access Memory*
- *USB* : *Universal Serial Bus*
- *VHSIC* : *Very High Speed Integrated Circuit*
- *VHDL* : *VHSIC Hardware Description Language*

## LISTA DE SÍMBOLOS

- $A$  : *Imagem analisada, onde é realizada a busca pela imagem de máscara.*
- $Q$  : *Imagem de máscara ou padrão a ser procurada na imagem analisada.*
- $m$  : *Quantidade de linhas radiais utilizadas no filtro Rafi*
- $n$  : *Quantidade de fatores de escala utilizados no filtro Cifi*

# **CAPÍTULO 1. INTRODUÇÃO**

## **1.1. INTRODUÇÃO**

Processamento de imagem e visão computacional tornaram-se populares em muitas áreas com aplicações em medicina, segurança, indústria e robótica. Muitos algoritmos poderosos continuam sendo desenvolvidos. Porém, são algoritmos que por sua vez, exigem cada vez mais processamento, com o uso intenso de cálculos matemáticos, tais como médias, interpolações ou correlações e outros ainda mais complexos.

Estes algoritmos experimentam assim, um contínuo aumento da complexidade em busca de maiores precisões e robustez, e têm ao mesmo tempo, requerido da infra-estrutura computacional, um desempenho cada vez maior. Sistemas capazes de realizar um grande número de cálculos matemáticos em alto desempenho, tendem a ser sistemas computacionais extremamente complexos e de alto custo.

Neste sentido, ao mesmo tempo em que algoritmos complexos de processamento de imagem necessitam apresentar os resultados com um desempenho o maior possível, a fim de viabilizar eventuais aplicações, ocorre uma demanda maior por sistemas computacionais capazes de oferecer tal desempenho com viabilidade competitiva.

Sistemas que utilizam tecnologia de processamento paralelo podem apresentar um desempenho semelhante ou mesmo superior aos sistemas computacionais convencionais para aplicações específicas, sendo uma opção de menor custo e também mais compacta.

Neste sentido, os dispositivos *Field Programmable Gate Array* (FPGA) representam uma tecnologia de processamento adequada e promissora, para muitos algoritmos de cálculos matemáticos repetitivos, devido principalmente à capacidade que os FPGAs têm, de realizar muitas tarefas em um único ciclo de relógio.

Um novo algoritmo de busca por imagem de máscara (*template matching*) invariante à rotação, escala, translação, brilho e contraste (RSTBC em inglês), chamado de Ciratefi foi recentemente proposto [1],[2]. Seu desempenho em computadores convencionais é competitivo em relação aos algoritmos para

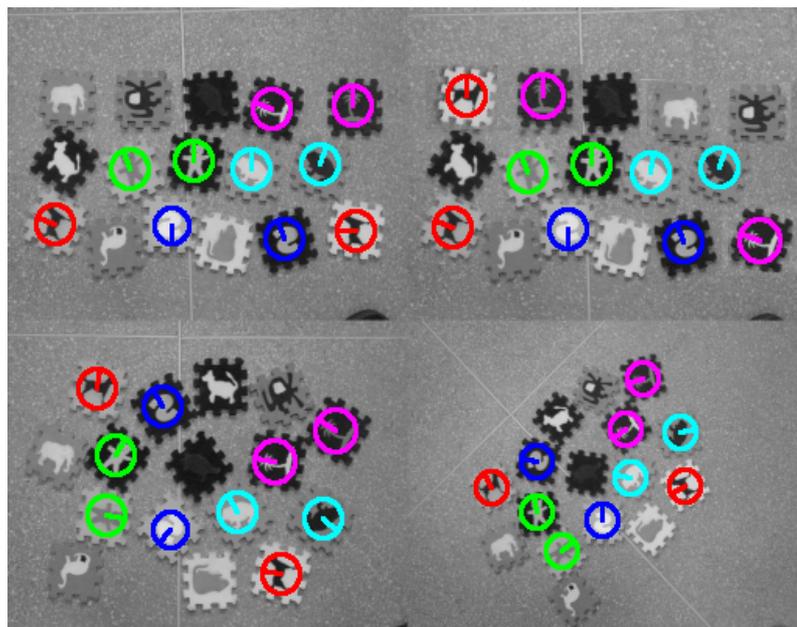
aplicações semelhantes, mas é da ordem de alguns segundos para imagens com resolução de 640x480.

O algoritmo Ciratefi é composto por três filtros que sucessivamente excluem os pixels que não possuem chance de pertencerem à imagem de máscara buscada (chamada de  $Q$  ou sub-imagem). Todos os filtros do Ciratefi, chamados de Cifi, Rafi e Tefi, varrem a imagem de maior resolução (imagem  $A$ ) onde é feita a busca pela sub-imagem  $Q$  e são todos invariantes à translação. O primeiro filtro é também invariante à rotação de  $Q$  e seleciona os pixels que possuem chance de pertencer à imagem buscada (pixels candidatos do primeiro grau), atribuindo-lhes o fator de escala mais adequado.

O segundo filtro processa radialmente os pixels candidatos disponibilizados pelo primeiro filtro. Os pixels que ainda possuem chance de pertencerem à imagem  $Q$  são elevados à condição de pixels candidatos do segundo grau. A estes pixels é atribuído o ângulo de rotação mais adequado.

O terceiro filtro é invariante ao brilho e contraste e responsável pelo casamento de imagens final.

A figura 1 demonstra o resultado final do algoritmo Ciratefi. Os pixels localizados estão indicados através de círculos com ponteiros radiais. Os círculos indicam a escala mais adequada para a figura de máscara encontrada na imagem. O raio a partir do pixel central indica a rotação mais adequada em relação à figura de máscara buscada.



**figura 1** Resultado final do Ciratefi com indicações de escala e rotação. Busca e localização independe também do brilho e contraste das imagens envolvidas.

O algoritmo Ciratefi repete a mesma série de operações para cada pixel, o que torna promissora sua implementação em *hardware* para execução em dispositivos de processamento paralelo. Entretanto, tal implementação em *hardware* do Ciratefi não é direta, pois seu correspondente sistema em *hardware* deve manter-se flexível às mudanças na resolução da imagem de máscara e no intervalo de escalas a serem processadas. Além disso, como no processamento em paralelo há muitos sinais sendo calculados ao mesmo tempo, é preciso gerenciar adequadamente as latências dos cálculos envolvidos e o sincronismo dos dados do sistema como um todo.

Neste trabalho, propomos um *software* que, dados os parâmetros de entrada, gera automaticamente os módulos compiláveis na linguagem de descrição de *hardware* VHDL, e que implementam o primeiro dos três filtros do Ciratefi em FPGA (filtro Cifi).

Os módulos gerados em VHDL estão otimizados e todos os seus processamentos estão sincronizados e registrados em *pipeline*. A performance de nossa implementação está otimizada na medida em que é capaz de classificar um pixel da imagem analisada, por ciclo de relógio, como *ocorrência* ou *não-ocorrência* na imagem de máscara procurada.

## 1.2. MOTIVAÇÃO

Visão computacional possui aplicações práticas em diversos campos do conhecimento e muitos trabalhos têm sido publicados com a implementação destes algoritmos em arquiteturas de processamento paralelo. São pesquisas relacionadas à visão computacional e que permitem aplicações desde à medicina de imageamento 3D [51], inspeções de qualidade [3], codificação [40] e sistemas autômatos [17], ou mesmo, aplicações espaciais[4]. O processamento paralelo é particularmente importante em algoritmos de processamento de imagem onde a mesma série de operações matemáticas deve ser repetida para cada pixel [6].

Muitos algoritmos poderosos de processamento de imagem têm sido propostos, tornando viáveis soluções automáticas e semi-automáticas para muitas tarefas populares em visão computacional, tais como o reconhecimento visual de objetos [5]. Mais especificamente, a localização de objetos em imagens em tons de cinza que sejam invariantes aos chamados parâmetros RSTBC é uma operação básica bastante útil para diversas tarefas de processamento de imagem e de visão computacional, tais como controle visual, registro de imagens e a computação de movimentos visuais [1].

As maiores dificuldades em algoritmos de detecção de objetos em imagens permanecem como sendo o custo computacional e a manutenção da robustez para distorções da imagem. Este último ponto pode ser atenuado pelo uso de tantas referências de imagens quanto possível, mas permanece ainda o problema do alto custo computacional [6]. Trata-se de algoritmos que utilizam operações em 2D, como FFT/IFFT, transformada polar e transformada de Hough que tornam ainda mais difícil realizar processamentos de localização de imagens em taxas de vídeo utilizando tais algoritmos [48].

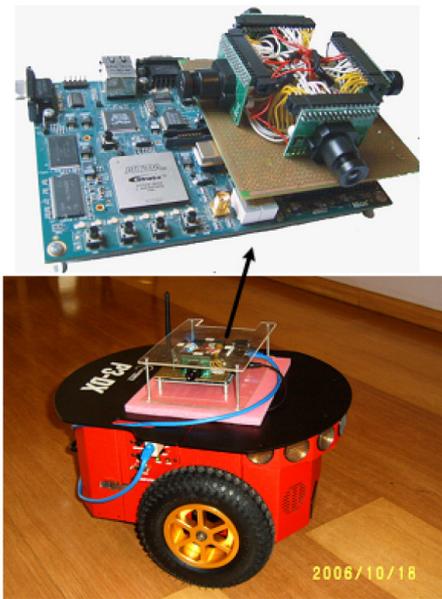
No entanto, outras tecnologias de processamento podem viabilizar estes tipos de soluções. Mesmo que os sistemas baseados em microprocessadores possuam taxas de relógio da ordem de 10 a 30 vezes superiores às dos FPGAs típicos, a grande quantidade de paralelismo espacial disponibilizada por modernos FPGAs, oferece um atraente potencial de aceleração de desempenho [7].

A vantagem do uso de dispositivos de lógica programável pode estar não só no ganho em desempenho mas também na possibilidade de diminuição de custos ou mesmo, no uso de processamento híbrido com sistemas em paralelo e outros tipos de processadores, com uma divisão harmônica entre as tarefas.

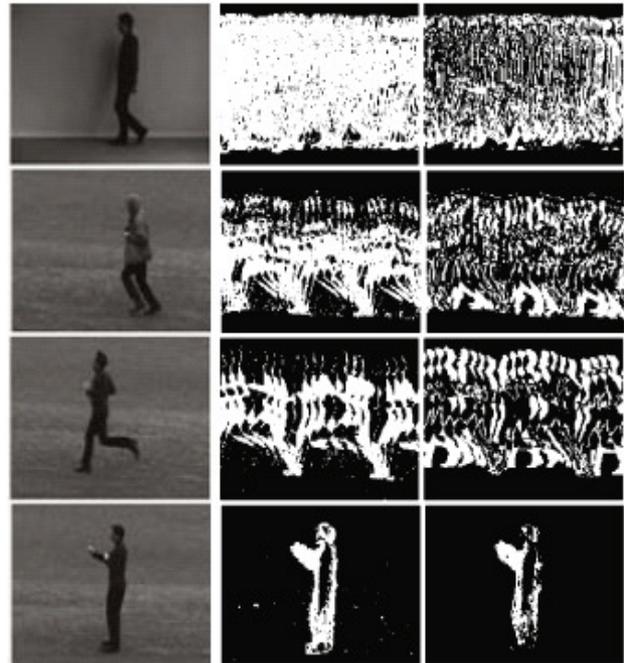
As figuras de 2 a 5 ilustram aplicações e trabalhos desenvolvidos em visão computacional e reconhecimento de padrões de imagem com a utilização de dispositivos FPGAs.



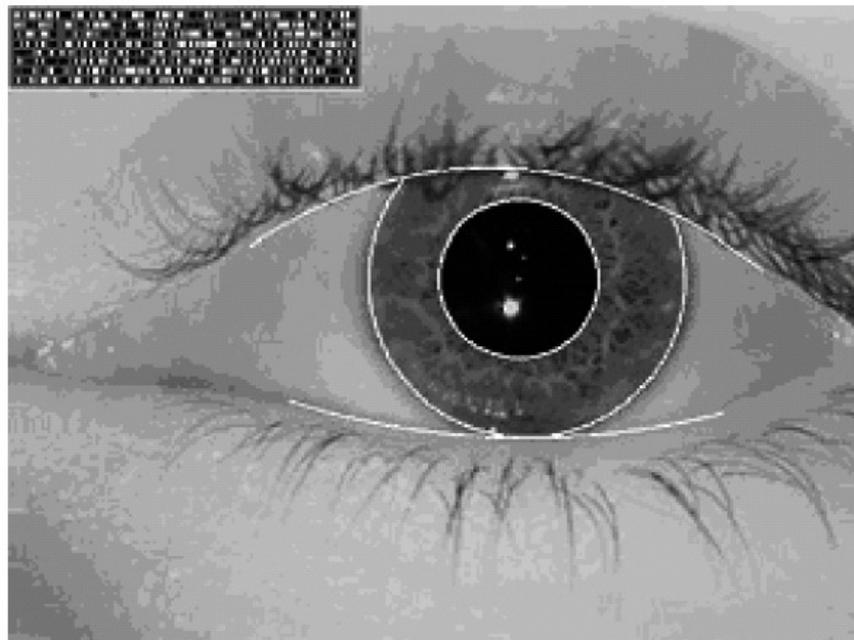
**figura 2** Exemplo da utilização do algoritmo SIFT sobre imagens estéreo obtidas pelo veículo *Spirit* em exploração sobre Marte [4].



**figura 3** Plataforma de movimentação autômata. Emprega sistema de reconhecimento de objetos com algoritmo SIFT em FPGA [17].



**figura 4** Imagens de Testes do sistema em FPGA de reconhecimento de ações humanas [42].



**figura 5** Área mensurável da íris para reconhecimento de padrões com implementação em FPGA. Imagem no alto à esquerda demonstra codificação em 2D da íris [40].

### 1.3. OBJETIVO

Nosso objetivo é propor o projeto de sistemas em *hardware* que viabilizem a aceleração expressiva no desempenho do algoritmo Ciratefi em comparação com sua execução em computadores convencionais. Sua implementação em *hardware*, com a obtenção de significativas melhoras no desempenho não é imediata e requer análises adequadas e criteriosas para manter uma elevada eficiência ao longo de toda a implementação.

Neste trabalho de mestrado, propomos uma solução em *hardware* para os dois primeiros filtros do algoritmo Ciratefi, mas implementamos e validamos apenas o filtro Cifi, o primeiro dos três filtros do Ciratefi. A solução para o filtro Rafi, o segundo filtro, é bastante similar. O filtro Tefi, o último filtro, é o mais rápido e sua implementação em *hardware* não fez parte da inovação proposta e pode permanecer como tarefa de *softwares* ou integrar sistemas de co-processamento discutidos mais adiante nesta dissertação.

Os requisitos mais importantes para o sistema proposto neste trabalho devem ser: alta flexibilidade para ser eventualmente usado por outros algoritmos similares de processamento; ser independente dos fabricantes dos dispositivos FPGAs; e finalmente, ser facilmente adaptável para diferentes parâmetros de entrada, mantendo elevado o nível de desempenho. Este último ponto é importante para permitir o uso do algoritmo Ciratefi em diferentes aplicações.

As estratégias desenvolvidas neste trabalho estiveram alinhadas com o objetivo de se conseguir o máximo desempenho no uso da tecnologia de processamento em paralelo dos dispositivos FPGAs. A concepção destas estratégias foi a etapa inicial do trabalho: a pesquisa e o desenvolvimento das técnicas mais apropriadas para as codificações segundo os requisitos do projeto.

Uma vez definidas as técnicas e as estratégias de programação, fizemos em uma segunda etapa, o projeto e a validação da metodologia proposta com a implementação do primeiro filtro Cifi do algoritmo Ciratefi para o *hardware*. A implementação do filtro Rafi é bastante similar, embora a quantidade de correlações a serem feitas seja mais elevada.

A solução proposta acelerou em aproximadamente 5000 vezes o tempo de execução para o processamento de um quadro completo do filtro Cifi de 7s (em um PC de 3GHz) para 1,367ms (em uma FPGA).

## 1.4. ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho está organizado da seguinte forma :

- Capítulo 2 - Revisão da Literatura. Este capítulo será dividido em duas partes distintas, uma fazendo a revisão teórica sobre algoritmos de busca e localização de imagem e outra apresentando os aspectos mais fundamentais do algoritmo Ciratefi além de uma análise de sua performance computacional.
  - 2.1 Fundamentação Teórica em Busca de Padrão em Imagem. Este item apresenta de forma sucinta os fundamentos e os principais algoritmos de busca e localização de padrão em imagens, além da importância e aplicações destes tipos de algoritmos.
  - 2.2. O Algoritmo Ciratefi. Neste item descrevemos o funcionamento do algoritmo Ciratefi. Mais especificamente os dois e mais complexos filtros de seleção de pixels . Apresentamos ainda as características de performance do Ciratefi em computadores convencionais.
- Capítulo 3. Tecnologia de Processamento. Este capítulo apresenta algumas das principais tecnologias de processamento, uma descrição sucinta da evolução dos dispositivos lógicos programáveis; a comparação entre processadores paralelos e os convencionais, além de informações mais detalhadas sobre o dispositivo de processamento usado neste trabalho. Finalmente, são apresentados os principais aspectos de programação para sistemas em *hardware* de alto desempenho.
- Capítulo 4. Métodos e Processos. Este capítulo apresenta os objetivos da implementação em *hardware* capaz de acelerar o desempenho do algoritmo Ciratefi. Em seguida, apresentamos as soluções, estratégias e métodos desenvolvidos para a resolução, implementação, análise e simulação do algoritmo Ciratefi em *hardware*, segundo os objetivos expostos.
- Capítulo 5. Resultados. Neste capítulo, estão descritos os resultados finais obtidos na implementação do primeiro filtro do algoritmo Ciratefi em *hardware* além de todos os módulos implementados. Estão também indicados os cálculos intermediários e finais, obtidos na simulação e implementação do algoritmo. Foram feitas análises comparativas de precisão entre os valores

obtidos nas diferentes etapas de simulação. Foi também analisado o desempenho do *hardware* implementado.

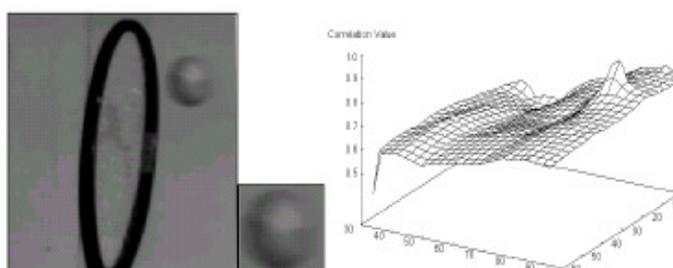
- Capítulo 6. Discussões. São discutidos e analisados os resultados encontrados, e propostas futuras alterações e otimizações possíveis para a arquitetura projetada, mesmo em termos de modelo de processador de lógica programável a ser utilizado.
- Capítulo 7 – Conclusão. Além das conclusões do trabalho realizado, são também apresentados os principais aspectos para uma infra-estrutura ideal de processamento, resolvendo o problema proposto com a eficiência máxima do *hardware* desenvolvido.

## CAPÍTULO 2. REVISÃO DA LITERATURA

O casamento de padrões é o processo de localização de uma sub-imagem, chamada de máscara ou padrão (*template*) dentro de uma imagem, de maior resolução. Os centros destas imagens de máscara encontradas podem ser usados como pontos de correspondência para determinação de parâmetros de registro. A busca e localização de máscaras envolve a comparação de uma dada máscara com janelas de processamento e a identificação da janela que é mais similar à máscara.[8] As estratégias gerais para a localização de máscaras envolvem a correlação como uma medida de similaridade entre as imagens [6].

A localização de padrão de imagens é uma das operações fundamentais em campos tais como, identificação de assinaturas em operações bancárias, reconhecimento de impressões digitais, inspeções visuais ou posicionamento de precisão. Nas décadas passadas muitos algoritmos de casamento de padrões foram propostos incluindo métodos como correlação normalizada, filtros de combinação [9], correlação tipo *phaseonly* [10], transformada de Hough-Fourier [11], e transformada geral de Hough [12] [48].

A busca por padrões em imagens é um método flexível e poderoso e pode ser usado ainda em aplicações de rastreamento de objetos com pequenas distorções de um quadro ao seguinte. Ao se adaptar as máscaras ao longo da seqüência de imagens, até mesmo grandes deformações podem ser ainda rastreadas. [13 Apud 14] A figura 6 demonstra um exemplo de utilização de rastreamento de objetos para apenas um quadro. À direita na mesma figura, há um gráfico em 3D das correlações entre a imagem e a sub-imagem (de máscara). O pico mais alto no gráfico de correlações, indica o melhor casamento.



**figura 6** Exemplos de busca e localização de imagem. Tem-se a imagem original à esquerda com a máscara em detalhe (centro).

A precisão do processo de busca e localização depende da precisão da métrica utilizada para determinar a similaridade entre a janela e a máscara. Quanto maior a precisão da métrica, mais acurado será o processo de busca e localização.

Diferentes métricas ou medidas de similaridade foram desenvolvidas. Não há uma única medida de similaridade que é conhecida por produzir o melhor resultado em todas as situações. Dependendo dos tipos de imagens utilizadas, uma medida de similaridade pode funcionar melhor do que outra. As principais medidas de similaridade são:

- Soma das diferenças absolutas
- Coeficiente de correlação
- Distância geométrica
- Informação mútua
- Momentos invariantes

Uma análise mais profunda e comparativa entre as técnicas atuais de casamento de padrões em imagens pode ser encontrada também em Kim et al. [1] e Kim et al. [2]. Na literatura há algumas outras implementações de busca e localização de imagens em FPGAs. Hegel et al. [15] apresentam um algoritmo em FPGA de busca e localização de imagem binária. Esta técnica não é invariante nem por escala nem por rotação. Shen et al. [16] apresentam um algoritmo de casamento de padrões de imagens invariante à rotação baseado em projeções circulares. Porém não é invariante à escala.

No trabalho [17] é implementado o algoritmo SIFT (*Scale Invariant Feature Transform*) em uma arquitetura de processamento paralela, utilizando cálculos matemáticos em ponto-fixe e uma versão para o *hardware* de cálculos trigonométricos CORDIC (*COordinate Rotation Digital Computer*). Segundo os autores do trabalho, trata-se do primeiro trabalho a implementar de modo completo o algoritmo SIFT em uma arquitetura FPGA. A preocupação dos autores com a otimização do *hardware* e em mantê-lo compacto o suficiente para ser implementado em uma única FPGA, resultou em um sistema completo de detecção em tempo real de busca e localização, com taxas de 30 quadros por segundo em imagens de 320x240. No sistema implementado, a flexibilidade no ajuste de parâmetros segundo diferentes aplicações é consequência da proposta inicial de projeto integrando *hardware/software*.

## 2.1. O ALGORITMO CIRATEFI

O objetivo do algoritmo Ciratefi é o de localizar uma imagem de máscara em tons de cinza  $Q$  em uma imagem de maior resolução a ser analisada  $A$ , invariante à rotação, escala, translação, brilho e contraste. Apresentamos nos próximos tópicos uma breve descrição do algoritmo. Para maiores detalhes é possível verificar a referência em Kim et al. [1].

O Ciratefi consiste em três filtros em cascata chamados Cifi, Rafi e Tefi. Cada filtro exclui sucessivamente pixels que não possuem chance de pertencer à imagem de máscara buscada.

O algoritmo Ciratefi mostrou-se um algoritmo de muito melhor desempenho do que o algoritmo que apresenta precisão similar e chamado de força bruta [1]. Ainda assim, mesmo com um desempenho cerca de 400 vezes superior ao do algoritmo de força bruta, o Ciratefi requereu cerca de 22 segundos para apresentar o resultado (em computador Pentium 4 de 3 GHz) da localização da máscara em imagens.

Os primeiros dois filtros, Cifi e Rafi são os mais lentos. O último filtro é cerca de 7 vezes mais rápido do que o primeiro filtro e 13 vezes mais rápido do que o segundo filtro. Neste sentido, este trabalho está focado na resolução dos dois primeiros filtros em *hardware*. Já o terceiro filtro integraria uma solução completa baseada em co-processamento FPGAs e processamento convencional.

### 2.1.1. O FILTRO CIFI

O primeiro filtro, chamado Cifi (filtro de amostragem circular), calcula a média dos pixels em tons de cinza das imagens  $A$  e  $Q$  em círculos e utiliza estas médias para quantificar a chance que os pixels nas duas imagens têm de serem correspondentes. Através deste parâmetro final (correlação), o filtro classifica determinados pixels de  $A$  como pixel candidato do primeiro grau, ou seja, pixels que possuem chance razoável de pertencerem à imagem de máscara. Este filtro também determina um *provável fator de escala* para cada pixel candidato.

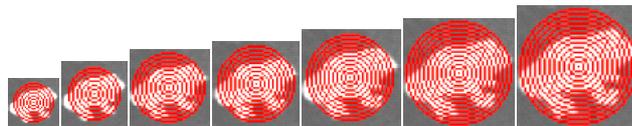
Para chegar a estes resultados, o filtro Cifi realiza sucessivas correlações entre a matriz 2-D,  $C_Q$  e a matriz 3D,  $C_A$ . A matriz  $C_Q$  é uma matriz pré-calculada,

antes do início do processo de localização e contém as médias dos valores dos círculos de  $Q$  em diversos fatores de escala figura 7).

Por sua vez, a matriz  $C_A$  é calculada para cada pixel da imagem  $A$  e contém, para cada pixel  $(x,y)$  em  $A$ , um vetor de valores de média dos círculos centrados em  $(x,y)$  obtidos através da equação 1.

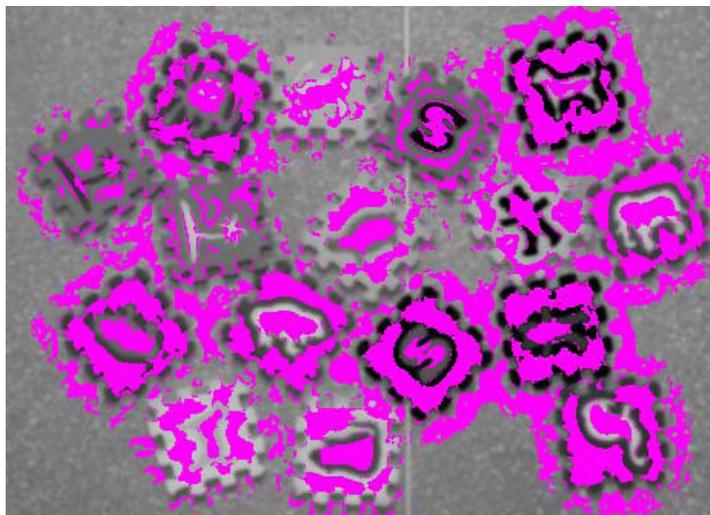
$$C_A[x, y, r] = \frac{1}{2\pi} \int_0^{2\pi} A(x + r \cos \theta, y + r \sin \theta) d\theta \quad (1)$$

Com a constituição da matriz 3D,  $C_A$  e o pré-processamento das diferentes escalas das imagens de máscara, é possível calcular os coeficientes de correlação e obter o melhor fator de escala para cada pixel candidato do primeiro grau. Um determinado pixel  $(x,y)$  é classificado como pixel candidato do primeiro grau se a melhor correlação calculada é maior do que um determinado valor de limiar especificado.



**figura 7** Projeções circulares para diferentes escalas.

A figura 8 apresenta o resultado visual do filtro Cifi, em que na imagem  $A$ , os pixels candidatos estão indicados em magenta. Para cada pixel candidato, há também a correspondente informação de qual das escalas é mais apropriada para cada pixel candidato.



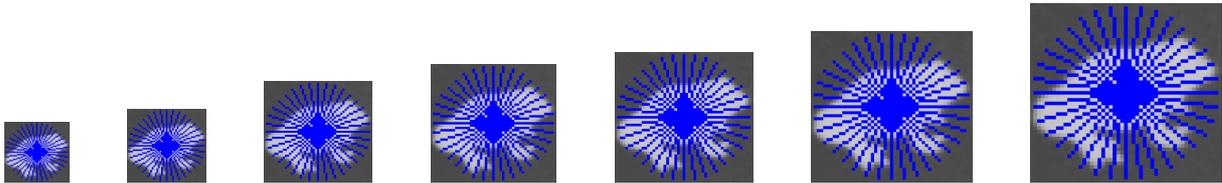
**figura 8** Resultado do filtro Cifi indicando em magenta os pixels candidatos do primeiro grau.

## 2.1.2. O FILTRO RAFI

O segundo filtro, chamado de filtro Rafi (filtro de amostragens radiais), calcula, para cada pixel candidato do primeiro grau  $(x,y)$ , as projeções das imagens de  $A$  em  $Q$  em linhas radiais (figura 9), com o raio  $\lambda$  dado pelo fator de escala computado por Cifi e onde as linhas radiais estão sendo projetadas para o cálculo das médias dos pixels segundo a equação 2.

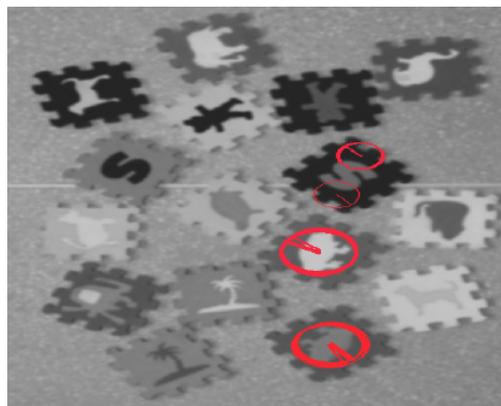
$$Ras_B^\lambda(x, y, \alpha) = \frac{1}{\lambda} \int_0^\lambda B(x + t \cos \alpha, y + t \sin \alpha) dt \quad (2)$$

O filtro Rafi atualiza os pixels candidatos do primeiro grau, que possuem chance de correspondência com a imagem de máscara, para pixels candidatos do segundo grau. O filtro realiza sucessivas correlações entre os dois conjuntos de projeções usando deslocamento circular. O filtro também calcula o *provável ângulo de rotação* para cada pixel candidato do segundo grau.



**figura 9** Projeção radial para a escala selecionada

A figura 10 apresenta o resultado do filtro Rafi sobre a imagem  $A$ , onde estão indicados os pixels candidatos do segundo grau. O círculo indica o mais provável fator de escala para o pixel no centro deste círculo (informação obtida através do filtro Cifi anterior) e o raio indica o mais provável ângulo de rotação em relação à imagem de máscara. Este ângulo é obtido pelo filtro Rafi.



**figura 10** O Resultado do filtro Rafi

### 2.1.3. O FILTRO TEFI

O terceiro filtro, chamado de filtro Tefi (filtro de casamento de máscara), é um algoritmo convencional de casamento de máscara, invariante ao brilho e contraste que é aplicado aos pixels candidatos do segundo grau, usando as escalas e os ângulos determinados respectivamente por Cifi e Rafi.

O filtro Tefi, cujo resultado esta exposto na figura 11, faz uso dos coeficientes de correlação para avaliar o quão próximo a máscara  $Q$  combina com o pixel candidato do segundo grau.

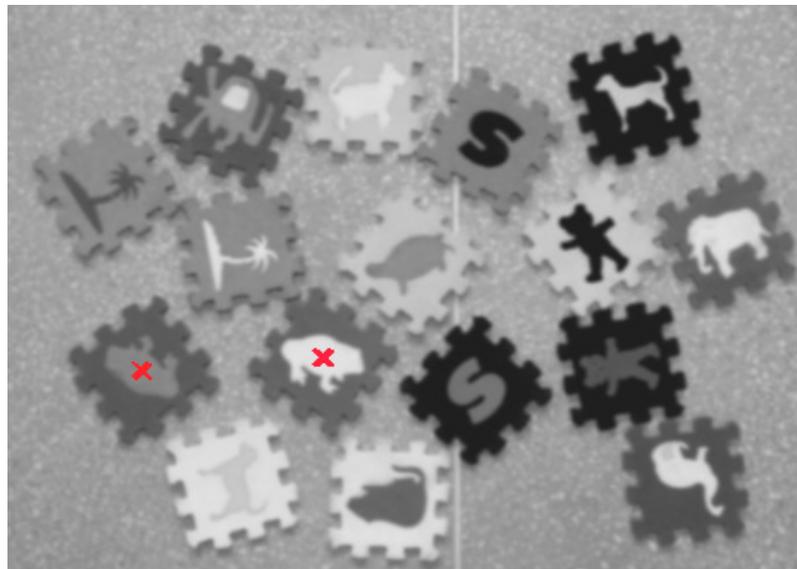


figura 11 Resultado do filtro Tefi indicando os pixels localizados.

### 2.1.4. PERFORMANCE DO ALGORITMO CIRATEFI

Os pontos interessantes do Ciratefi são sua robustez e acuracidade quando comparado com outros algoritmos [1]. Entretanto, o algoritmo Ciratefi requer vários segundos para calcular as posições de combinação e localização.

Para o pior caso testado (com a imagem  $A$  com resolução de  $465 \times 338$ , a imagem de máscara  $Q$  com resolução de  $52 \times 51$  pixels, 6 escalas e 36 ângulos) o processamento completo do Ciratefi requereu 22s, divididos da seguinte forma:

- O primeiro filtro – Cifi, requereu 2.5s para o cálculo da matriz 3D,  $C_Q$  e 4.5s para o processo de cálculo das correlações, em um total de 7s;
- O segundo filtro – Rafi, requereu 13s para apresentar o resultado;
- O último filtro – Tefi, foi o mais rápido e requereu cerca de 1s para apresentar um resultado.

Estes tempos foram obtidos usando um computador Pentium 4 de 3GHz. Os cálculos mais complexos são aqueles envolvidos com a computação da correlação. Os filtros Cifi e Rafi são muito similares no que se refere à matemática envolvida. O que torna o filtro Rafi mais lento do que o Cifi é justamente a maior quantidade de correlações, por pixel da imagem  $A$ , que o filtro Rafi deve fazer (36 ângulos diferentes) em relação ao filtro Cifi (6 escalas diferentes).

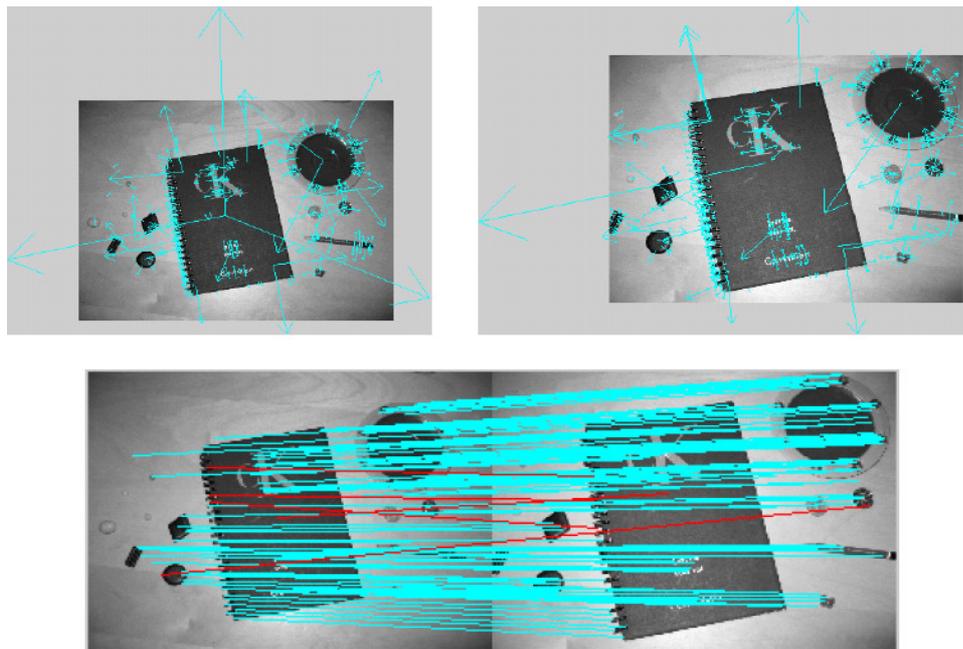
## 2.2. O ALGORITMO SIFT

Em Beukelman et al. [70] é exposto um breve histórico do surgimento de algoritmos de casamento de imagens. Um dos primeiros algoritmos data de 1988 e foi proposto por Harris e Stephens [18].

A partir de diversas contribuições, tem havido um esforço contínuo para desenvolver algoritmos robustos para detectar características invariantes às transformações de rotação, escala, e mudança de iluminação. Entre as propostas, Lowe [19] apresentou uma das mais populares, chamada de *Scale-Invariant Feature Transform* (SIFT). O Sift utiliza como base, a distribuição do gradiente de pequenas regiões em diferentes escalas para determinar um conjunto de pontos-chave que permitem identificar objetos.

As principais etapas envolvidas no algoritmo Sift são: a detecção de extremos escala-espço, a localização de pontos-chave, a verificação da orientação e o descritor para os pontos-chave. Maiores análises sobre o algoritmo Sift podem ser encontradas também em Andersen et al. [20]. A figura 12 apresenta imagens de teste, processadas pelo algoritmo SIFT.

O algoritmo Sift não é tratado neste trabalho. A implementação detalhada do algoritmo Sift em *hardware* pode ser encontrada em Bonato [17].



**figura 12** Imagens ilustrando o processamento pelo algoritmo Sift. Os pontos-chave são ilustrados como vetores indicando a localização, escala e orientação [20].

## **CAPÍTULO 3. TECNOLOGIA DE PROCESSAMENTO**

Neste capítulo será feito um breve histórico do surgimento dos dispositivos de lógica programável, dos primeiros até os mais modernos FPGAs. Embora o *hardware* implementado neste trabalho possa ser configurado para FPGAs de qualquer fabricante, (com tamanhos lógicos adequados) foi preciso selecionar uma FPGA apropriada para os ensaios de desempenho. Pretende-se ainda mostrar as principais características da FPGA selecionada para os ensaios e simulações dos módulos em *hardware* gerados neste trabalho.

### **3.1. DISPOSITIVOS DE PROCESSAMENTO PARALELO**

A evolução na fabricação de sistemas digitais tem sido enorme nas últimas décadas. Até a primeira metade do século XX utilizavam-se válvulas para implementar qualquer circuito eletrônico. Nos anos 50 surgiram os transistores, e até a década de 60, os sistemas digitais eram construídos a partir de componentes discretos, tais como transistores, diodos e resistores.

Com o surgimento dos circuitos integrados (CIs), passou a ser possível integrar inicialmente alguns poucos e atualmente milhões, de transistores em uma única pastilha de silício [21]. Apesar de inicialmente a tecnologia permitir somente uma pequena integração, atualmente, com a evolução desta tecnologia, é possível integrar 14 milhões de transistores por  $\text{cm}^2$ , podendo esta densidade atingir 100 milhões até o ano de 2012 [22].

Este avanço permitiu que uma grande variedade de circuitos digitais e analógicos fosse implementada em pastilhas de silício. A maior densidade em transistores por  $\text{mm}^2$  é encontrada em circuitos digitais, onde a tecnologia atual permite implementar comercialmente circuitos de FPGA com geometria de 65nm [23] ou até 40nm [24].

Como normalmente os circuitos integrados desempenham operações pré-definidas pelo fabricante, para a elaboração de um circuito complexo com função

definida pelo usuário, há necessidade do uso de tipos diferentes de circuitos integrados, trazendo diversas desvantagens ao sistema, tais como [21]:

- aumento da área de placa necessária para a conexão dos diversos componentes que fazem parte do sistema;
- perda de desempenho devido ao atraso de roteamento na placa;
- aumento do consumo de energia.

Para contornar estes problemas, surgiram outros tipos de circuitos integrados que possuem as funcionalidades definidas pelo usuário e não mais pelo fabricante, são os dispositivos programáveis.

### **3.1.1. DISPOSITIVOS TIPO ASIC**

Os dispositivos ASIC (*Application Specific Integrated Circuit*) são circuitos integrados programáveis. Esta tecnologia permite que engenheiros implementem algoritmos sem a necessidade de compreender a física dos processos de fabricação de semicondutores. Isto é possível porque os fabricantes de circuitos ASIC oferecem uma biblioteca de células e funções que o projetista pode usar, dispensando a necessidade de conhecer como tais funções são implementadas em silício [25]. O fabricante oferece ainda diversas ferramentas e programas para projeto, compilação, síntese, roteamento e simulações.

Após a etapa de projeto, o fabricante do circuito integrado se responsabiliza por todas as etapas de fabricação em silício do dispositivo ASIC projetado: ou seja, o projeto de fabricação, de leiaute, criação de máscaras e manufatura.

Os dispositivos ASIC são divididos em *Gate Array*, *Structured ASIC* e *Standard Cell*, onde o projeto é baseado em blocos pré-definidos, constituídos por portas lógicas (em geral NAND), blocos básicos pré-definidos ou células complexas prontas, como *CPUs (Central Processing Unit)*, blocos de memória, *PLLs (Phase-Locked Loops)* e outros [26].

Toda a conexão dos transistores é completamente determinada pelo projeto que se está implementando. Uma vez terminado o projeto, o programa de análise e leiaute determina quais dos transistores serão conectados. Os dispositivos ASIC são fabricados com uma arquitetura tal que, uma vez determinado o projeto do circuito pelos usuários da tecnologia, o fabricante precisa apenas acrescentar a última

camada metálica que conecta as portas lógicas, utilizando os processos usuais de fotolitografia para cada camada metálica [25].

### **3.1.2. MEMÓRIAS PROGRAMÁVEIS DE LEITURA**

As memórias PROM (*Programmable Read Only Memory*) são memórias simples e programáveis pelo usuário para conter algum padrão específico, o qual pode ser usado para representar e armazenar um programa de um microprocessador, um simples algoritmo ou uma máquina de estado. Algumas PROMs podem ser programadas apenas uma vez, outras como EPROMs ou EEPROMs podem ter seu conteúdo apagado ou escrito diversas vezes.

PROMS são excelentes para implementar qualquer tipo de lógica combinatória, com um número limitado de entradas e saídas. No caso de lógicas seqüenciais, dispositivos de relógio externos devem ser utilizados. Este tipo de memória no entanto, tende a ser extremamente lenta.

### **3.1.3. DISPOSITIVOS PLA**

Os dispositivos PLA (*Programmable Logic Arrays*) foram uma solução para as limitações de velocidade e de sinais de entrada das PROMs. Os PLAs consistem de uma grande quantidade de entradas conectadas a um plano do tipo E, onde diferentes combinações de sinais podem ser processadas logicamente com o plano E, de acordo com a programação. As saídas da combinação com o plano E por sua vez, são direcionadas a um plano OU, onde podem ser mais uma vez processadas logicamente para finalmente serem direcionadas à saída do dispositivo. Nas entradas e saídas dos dispositivos pode ainda haver inversores de modo que a lógica NEGADO possa também ser obtida [25].

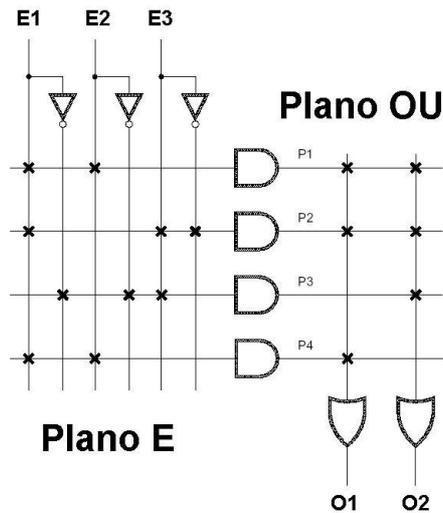


figura 13 Arquitetura de um dispositivo PLA [25].

A saída de cada célula é uma função booleana das entradas, conforme pode ser visto na figura 13. A densidade destes dispositivos é baixa (até 600 portas equivalentes), o tempo de propagação alto ( $> 10$  ns) e o consumo também é considerável [21].

### 3.1.4. DISPOSITIVOS PAL

Os dispositivos PALs (*Programmable Array Logic*) foram a primeira geração comercial de sucesso dos dispositivos lógicos programáveis [21] e são uma variação dos PLAs que como aqueles, possuem um amplo plano E para processar as entradas. Entretanto, o plano OU é fixo, o que limita o número de termos que pode passar por um processamento lógico do tipo OU. Com um diferencial, os dispositivos PALs possuem outros circuitos lógicos como multiplexadores, portas OU-exclusivo e registradores que podem ser adicionados às portas de entrada e saída do sistema. Mas o mais importante foi a inclusão dos circuitos de relógio, tipicamente *flip-flops*, que permitem agora, implementar uma grande quantidade de funções lógicas seqüenciais, necessárias para a implementação de máquinas de estado. A figura 14 ilustra a arquitetura de um dispositivo PAL e demonstra a flexibilidade de operações lógicas alcançadas com este tipo de dispositivo.

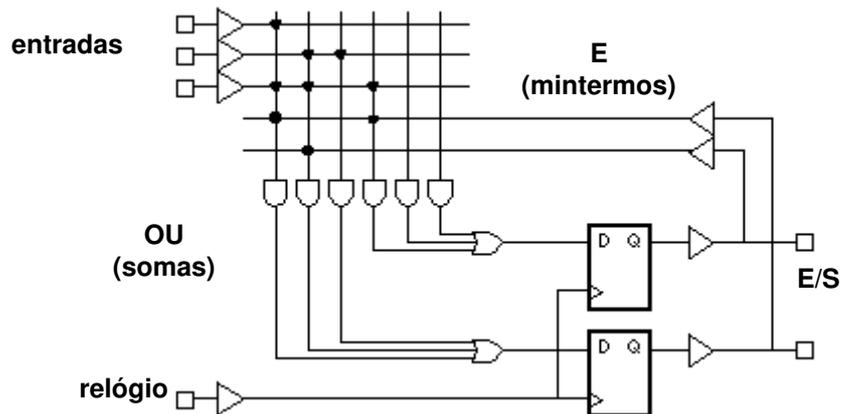


figura 14 Arquitetura de um dispositivo PAL [25].

### 3.1.5. DISPOSITIVOS CPLD E FPGA

O dispositivo lógico ideal para os projetistas era antes visto como aquele que oferecesse a flexibilidade e complexidade de um ASIC e tivesse a mesma facilidade de implementação de um dispositivo programável. A solução veio na forma de dois novos dispositivos: as CPLDs (*Complex Programmable Logic Device*) e as FPGAs (*Field Programmable Gate Array*).

#### 3.1.5.1. CPLD

Na década de 80 surgiram as primeiras *CPLD* comerciais. As principais inovações introduzidas pelas *CPLDs* foram [21]:

- A inclusão de uma matriz de conexões que permite que sinais de saída de um bloco lógico sirvam de entrada para outros blocos lógicos com a conexão entre estes blocos sendo feita internamente no dispositivo;
- A inserção de sinais globais, que permitem que alguns sinais sejam enviados a todo o dispositivo com atraso mínimo ;
- Uma arquitetura que permita que qualquer pino do dispositivo possa ser programado como pino de entrada, de saída ou de entrada-e-saída.

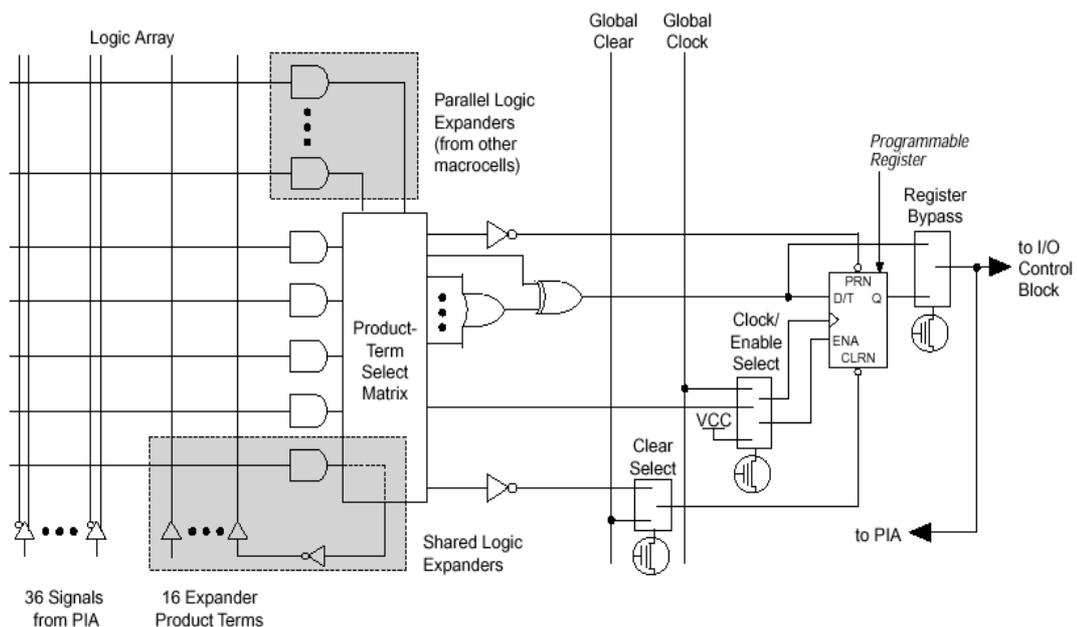
Dispositivos CPLDs podem ser vistos essencialmente como uma grande quantidade de dispositivos PALs, mas em um único circuito integrado, conectados entre si através de uma chave de ponto em cruz. Tanto os dispositivos PALs quanto

os CPLDs são baseados na mesma tecnologia e são usadas as mesmas ferramentas de desenvolvimento e programação. Os dispositivos CPLDs no entanto, permitem a utilização de uma lógica muito mais complexa além de possuírem uma maior capacidade de tais lógicas.

Embora os CPLDs de cada fabricante possuam diferentes variações, no geral todos se assemelham pois são constituídos por blocos funcionais, blocos de entrada e saída, e uma matriz de interconexão. Os dispositivos são programados usando-se elementos que, dependendo da tecnologia do fabricante, podem ser células EPROM, células EEPROM ou Flash EPROM.

A interconexão em um CPLD é uma matriz muito grande de chaves que permite aos sinais de várias partes do dispositivo se dirigirem para todas as outras partes do dispositivo. Mesmo que nenhuma chave possa conectar todos os blocos internos de funções entre si, ainda há flexibilidade capaz de permitir muitas combinações de conexões [25].

Na figura 15 vemos a arquitetura básica de uma CPLD da família MAX3000A [ 27 ] da Altera [ 28 ]. Pode-se verificar na arquitetura apresentada a lógica combinatória, composta pela lógica E programável, pela lógica OU fixa, incluindo termos de produto (lógica E), e por uma lógica OU-EXCLUSIVO adicionada após a lógica E, que permite inverter ou não a saída da lógica E [21].



**figura 15** Diagrama de uma macrocélula de CPLD MAX3000A da Altera [21].

### 3.1.5.2. FPGA

A tecnologia de matriz de portas programáveis é uma tecnologia relativamente nova, introduzida em 1984 pela Xilinx [29]. Trata-se de uma matriz 2D de células lógicas CLB (*Configurable Logic Blocks*) que podem ser arbitrariamente conectadas em uma rede de interconexão interna. Tanto as funções das células quanto as conexões são livremente programáveis no carregamento do programa de configuração no dispositivo, o chamado *bitstream* [14].

A inovação introduzida pelas FPGAs foi o número indefinido de ciclos de reconfiguração (cada um com duração de milisegundos dependendo da tecnologia utilizada de memória não-volátil externa ao FPGA) e a alta complexidade. A arquitetura de uma FPGA consiste de dois elementos básicos, o plano funcional e o plano de programação.

O plano de programação é invisível para o usuário e implementa fisicamente a configuração da FPGA. Este plano consiste ainda de um elevado número de elementos de armazenamento simples, cada um controlando a função de certa parte do plano funcional. Durante a configuração, todos elementos são carregados com um bit de configuração do *bitstream*.

A implementação mais importante do plano de programação são as células SRAM conectadas a um registrador de deslocamento muito grande. Esta implementação permite maior velocidade na configuração e um número indefinido de ciclos de configuração. Implementações alternativas são feitas através de *antifuses* restritos para uma única configuração e EEPROMs que permitem cerca de 1000 ciclos de configurações com velocidades extremamente reduzidas.

A arquitetura mais interessante presente nas células lógicas são as LUTs (*Look-up Tables*) que permitem a implementação de funções booleanas arbitrárias tendo normalmente de 4 a 6 entradas. FPGAs modernas possuem vários milhares de células lógicas. O terceiro elemento do plano funcional são as células de entrada e saída (E/S) que conectam a FPGA ao mundo exterior. Cada célula de E/S é programada como entrada, saída ou como um pino bidirecional [14].

As FPGAs [25;22;30] são assim chamadas porque apesar de terem uma estrutura similar a dos dispositivos PAL ou outro dispositivo programável, sua estrutura ainda se assemelha mais a de uma matriz de portas lógicas tipo ASIC. Isto

faz com que as FPGAs sejam ideais na substituição de dispositivos ASIC em aplicações que necessitem chegar rapidamente ao mercado, em detrimento do custo. Faz parte da estratégia no entanto, com o produto já no mercado, substituir as FPGAs pelos dispositivos ASIC para minimizar o custo a médio e longo prazo.

Como as FPGAs são dispositivos lógicos programáveis baseados em tecnologia SRAM, necessitam de configuração ao serem ligados. Para tal, usa-se um dispositivo de memória não volátil externo para armazenar a configuração inicial do dispositivo.

Nestes dispositivos, cada bloco lógico é composto por um registrador e uma tabela do tipo *Look-Up Table* que na maioria dos dispositivos FPGAs é implementada como um bloco de memória de 16 bits internos, 4 bits de entrada (endereço) e um bit de saída. Toda implementação de lógica combinatória (exceto a implementação de multiplexadores) é feita utilizando as LUTs. Ao cascatear várias LUTs, pode-se implementar lógica combinatória com a complexidade desejada.

A figura 16 apresenta um diagrama da arquitetura básica de uma FPGA, indicando as estruturas físicas com funções específicas de sincronismo e distribuição de relógios, lógica, memória e pinos de entrada e saída.

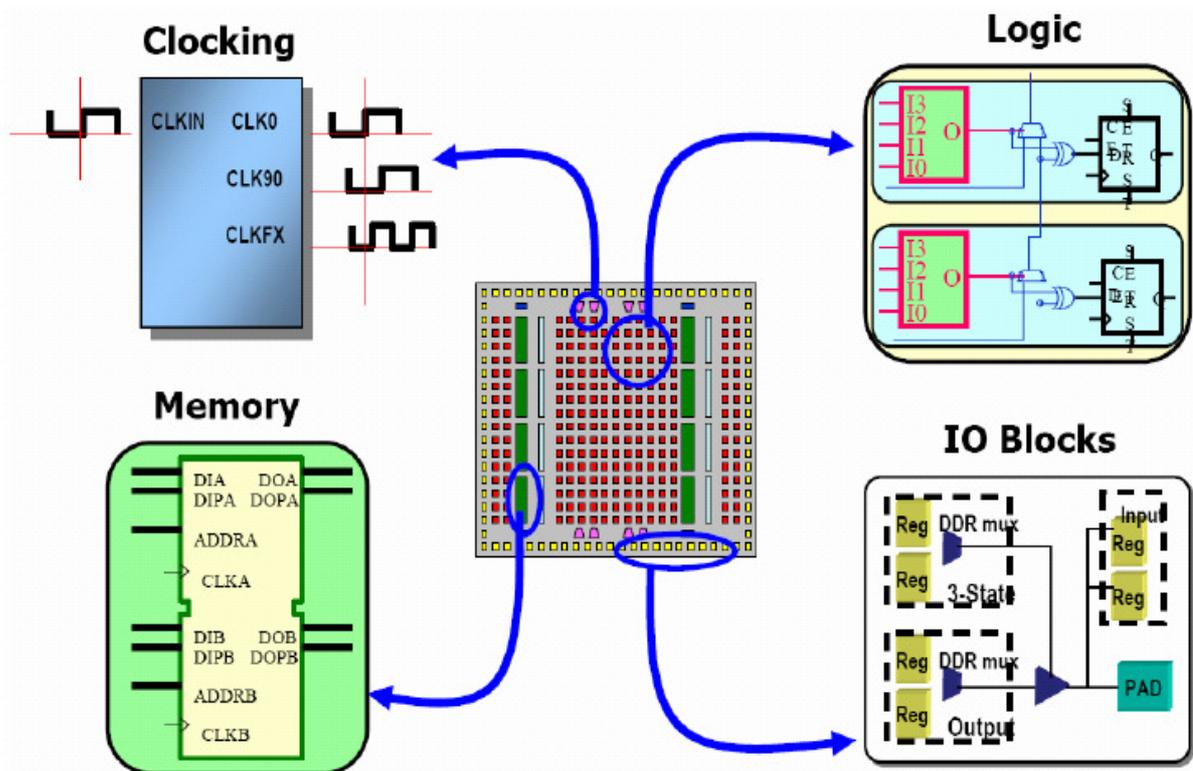
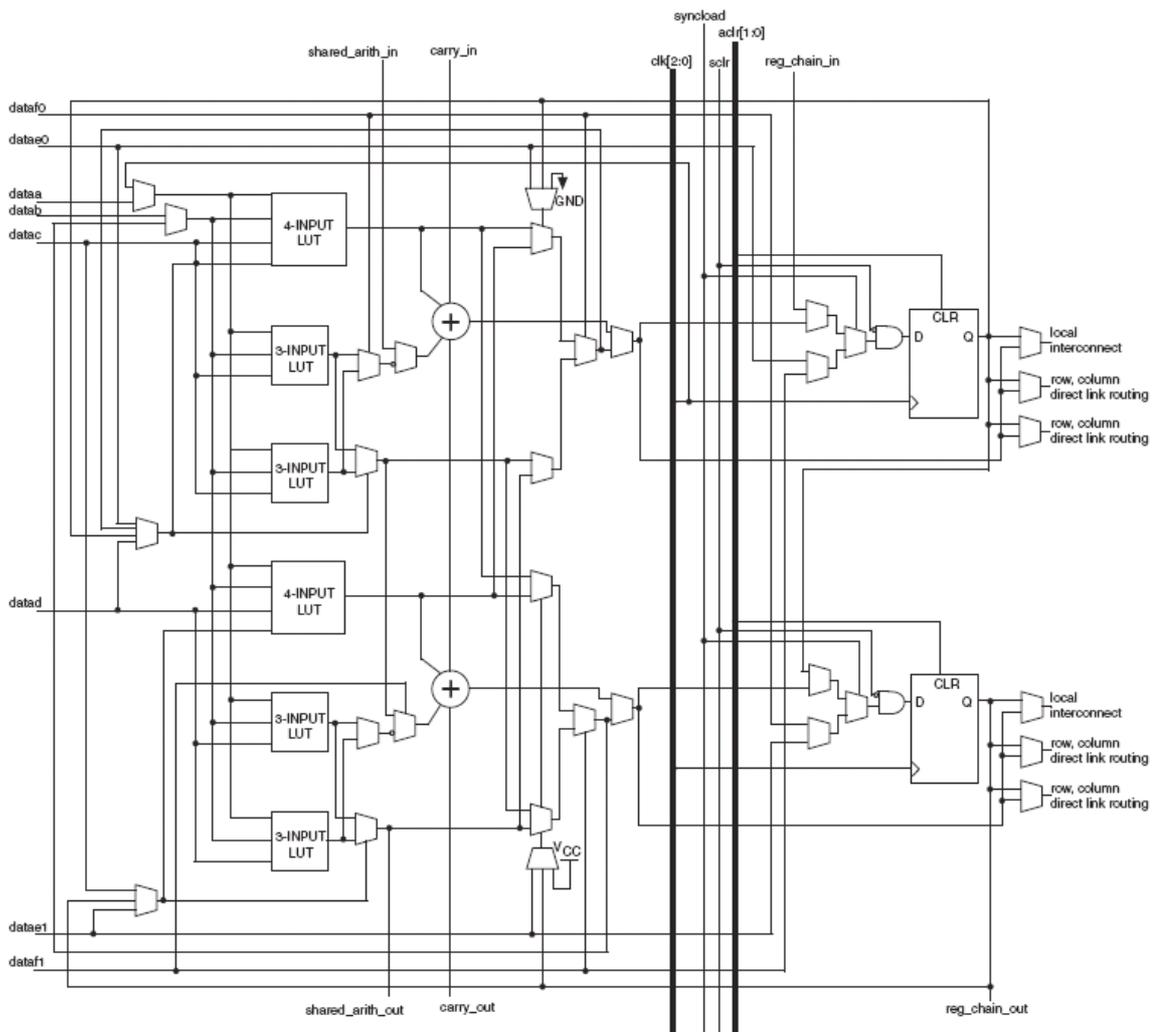


figura 16 Diagrama da arquitetura básica para uma FPGA [31].

A configuração inicial das LUTs é feita pelo programa de desenvolvimento, que cria a tabela de configuração das LUTs e as interliga, permitindo assim a implementação especificada no projeto.

A arquitetura das FPGAs consiste assim, de blocos configuráveis de lógica, blocos configuráveis de entrada e saída, e finalmente, de interconexões programáveis. Além disso, as FPGAs possuem ainda circuitos de relógio para distribuírem sinais de relógio para cada bloco lógico. Há também recursos adicionais que podem estar disponíveis como ALUs, memórias e decodificadores. Na figura 17 tem-se o diagrama para apenas um elemento lógico de um dispositivo FPGA da família Stratix III [32], da Altera.



**figura 17** Diagrama de um elemento lógico de uma FPGA Stratix III da Altera [32].

## 3.2. PRINCIPAIS DISPOSITIVOS DE ACELERAÇÃO DE PROCESSAMENTO

O elevado número de pixels em imagens digitais opera como um fator de aumento da complexidade para muitos algoritmos de processamento de imagens. Em particular, imagens 2D ou 3D de alta resolução ou seqüências em movimento requerem um enorme desempenho computacional. Combinada com requisitos de processamento de imagem em tempo real, há um aumento significativo na demanda por processamento nos sistemas eletrônicos físicos de alto desempenho.

Nos tópicos seguintes são discutidas brevemente três tecnologias de dispositivos de alto desempenho.

### 3.2.1. DSPs

Um processador digital de sinais (DSP) é similar aos processadores de propósito geral (GPP) em muitos aspectos. Eles possuem lógica física, isto é, as conexões lógicas entre as portas (*gates*) não podem ser modificadas. Eles oferecem aos programadores um conjunto de instruções fixas (ISA). E este conjunto de instruções é executado em modo seqüencial (em oposição ao fluxo de processamento dirigido aos dados). A maioria dos DSPs apresenta uma estrutura completa com instruções aritméticas e lógicas, acesso à memória, registradores e controle de fluxo [34].

Atualmente a tendência em processamento de imagem é em direção ao uso de *hardware* de propósito geral. A arquitetura dos GPP, entretanto, é muito pouco adequada para a estrutura de algoritmos de processamento de imagem. Em um GPP, instruções são processadas seqüencialmente e sua estrutura é otimizada para operações complexas em grandes operandos. Por este motivo, microprocessadores exploram certo nível de paralelismo e DSPs dedicados possuem melhoras consideráveis em relação aos processadores padrão.

Diferentemente de CPUs de propósito geral, um conjunto de instruções DSP é otimizado para operações com matrizes, particularmente multiplicação e acumulação (MAC) tradicionalmente em aritmética do tipo ponto-fixa, embora existam DSPs com

aritmética do tipo ponto flutuante. DPSs exibem estruturas em *pipeline* e por isso experimentam fluxo de programas bastante lineares sem saltos condicionais [34].

Como os DPSs normalmente executam pequenos programas com enormes quantidades de dados ou em pacotes tipo *streaming*, ambos tipos de informações são armazenadas em blocos de memória, freqüentemente acessíveis por barramentos em separado. Esta arquitetura é chamada de arquitetura *Harvard*, em oposição a arquitetura *Von Newmann* dos GPPs, onde tanto os dados quanto o programa são armazenados na mesma memória [34].

Há vários DPSs de alto desempenho disponíveis, incluindo aqueles com multi-núcleos com CPU de propósito geral em um mesmo circuito integrado. Em particular, há um interesse especial para os DPSs especializados em processamento de imagem e vídeo, também conhecidos como *media-processor*.

Ferruz e al. [33] desenvolveram um método de rastreamento de imagens baseado em um módulo de visão computacional em tempo real. Neste método, o desempenho de processamento em tempo real é atingido através da implementação de um sistema baseado em DSPs de multiprocessadores em paralelo.

Entre os fabricantes de DSPs estão Agere Systems, Analog Devices, Infineon, Lucent Technologies, Equator Technologies, Freescale Semiconductor (parte da Motorola), NXP (parte da Philips Electronics), Texas Instruments, e Zilog. Muitos destes fabricantes oferecem placas de desenvolvimento específicas para processamento de imagem e visão computacional, com câmeras e *softwares* [34].

### 3.2.2. GPUs

A padronização das interfaces gráficas em computadores demorou muito para ocorrer. Embora a necessidade de tal padronização date da década de 1970, foi apenas nos anos 90 que a SGI e a Microsoft publicaram seus respectivos padrões OpenGL e Direct3D. O efeito nas indústrias de *hardware* e *softwares* gráficos foi imenso. Os processadores especiais desenvolvidos para a renderização gráfica, chamados de *Graphics Processing Unit* (GPU) ultrapassaram a densidade de integração dos transistores dos processadores *General-Purpose Processor* (GPP).

Muitas operações gráficas mostraram-se muito bem adaptadas para o processamento em paralelo. Isto permitiu o desenvolvimento de unidades de processamento em paralelo nas GPUs, chamados de *shaders*. As mais recentes GPUs possuem centenas de unidades de processamento tipo *shader* que podem realizar operações em diferentes conjuntos de dados, produzindo resultados independentes. Em contraste com dispositivos DPSs ou CPUs similares, na GPU há um enorme cuidado no modo de sincronizar e agendar as tarefas, assumindo, entretanto, que a execução destas tarefas é altamente independente uma das outras.

Os assim chamados programas tipo *shader* são códigos específicos e devem respeitar certas restrições. Enquanto especificações mais recentes permitem certo grau de relaxamento destas restrições, tradicionalmente os códigos não possuem instruções de ramificações dinâmicas, são limitados às operações em ponto-flutuante e não podem exceder 100 instruções (ou outro parâmetro fixo).

Programas tipo *shader* são ideais para vértices ou fragmentos, efetuando diversos cálculos para determinação de características geométricas no espaço, envolvendo cores no caso dos vértices ou cálculos de textura em 2D no caso dos fragmentos.

O modelo de programação para *shaders* é dirigido aos dados e para se utilizar os GPUs e *shaders* para tarefas como processamento de imagem ou visão computacional, os dados devem ser empacotados para serem tratados como dados gráficos, e as operações devem também ser adaptadas para operações gráficas. Entretanto, têm surgido linguagens de programação específicas que fazem estas adaptações automaticamente, simplificando o processo de conversão. Alguns exemplos são a Cg da empresa Nvidia, o HLSL da Microsoft e mais recentemente o

ambiente CUDA (*Computer Unified Device Architecture*) da Nvidia que tende a expandir as aplicações dos GPUs para além dos objetivos gráficos mais imediatos. O ambiente possui uma linguagem baseada em C e um compilador oferece bibliotecas e tarefas de cálculos matemáticos.

Os GPUs oferecem processamento intenso de dados para aplicações que requeiram tempo-real. Embora estes processadores tenham sido tradicionalmente desenvolvidos para aplicações gráficas de renderização, eles são essencialmente multiprocessadores em paralelo que podem manipular cálculos matemáticos de modo bastante eficiente. Além disso, as GPUs são otimizadas para cálculos em ponto-flutuante, em contraste com DSPs, otimizados para cálculos com inteiros.

Neste sentido, há muita pesquisa sendo feita com o objetivo de utilizar os GPUs como co-processadores em visão computacional e outras aplicações além das aplicações gráficas. Projetistas devem no entanto, ainda pesar questões como desempenho e consumo, uma vez que as GPUs tendem a demandar recursos de alto consumo de potência. A fim de tomar de modo completo os benefícios de implementações com GPU, os projetistas devem particionar adequadamente seus algoritmos para os GPUs [34].

### **3.2.3. FPGA**

Alto desempenho também está disponível utilizando circuitos integrados de propósito especial, os dispositivos ASIC, os quais são otimizados para certas tarefas de processamento de imagem, usualmente explorando o paralelismo completo e inerente da tecnologia. Ao ajustar o barramento interno para o número de bits requerido, o silício de um ASIC é usado de forma muito eficiente. Campos típicos de aplicação são placas gráficas de vídeo, aceleradores 3D, ou circuitos integrados MPEG.

O enorme poder computacional das ASICs é acompanhado no entanto, de sérias desvantagens. A maior delas é a falta de flexibilidade devido à restrição de uma única funcionalidade. Além disso, o desenvolvimento de um ASIC é bastante caro e demorado. Atualizações sempre requerem re-projetos. Por estas razões, apenas um conjunto limitado de funções importantes é normalmente implementado em ASICs, o que leva a encontrá-los somente em aplicações de massa.

Por outro lado, os dispositivos FPGAs são uma alternativa muito interessante, pois combinam a alta velocidade de *hardwares* de propósitos especiais com elevada flexibilidade.

As FPGAs consistem de um elevado número de células lógicas simples, que podem ser conectadas ao se carregar um programa de configuração no dispositivo. Uma FPGA pode ser reconfigurada em milisegundos, com um novo *hardware* com aplicação muito diferente da configuração anterior.

FPGAs possuem um grande recurso de paralelismo e oferecem muita flexibilidade em sistemas de processamento de imagem embarcados [34]. Algumas características que, em geral, tornam atraentes os dispositivos de processamento paralelo aplicados em processamento de imagem é a própria estrutura dos algoritmos, que usualmente, permite um elevado grau de paralelismo e o baixo número de bits requeridos por pixel [14].

Por outro lado, grandes FPGAs possuem elevado consumo de energia e as taxas de relógio são menores do que as taxas de DSPs típicos. Uma grande variedade de dispositivos FPGA está disponível atualmente no mercado. Um aspecto importante na escolha do dispositivo FPGA mais adequado para a aplicação em visão computacional é o critério de memória interna, visto que normalmente as FPGAs possuem uma quantidade de pinos abundante comparativamente a capacidade lógica interna.

Interfaces com memória interna à FPGA oferecem acesso irrestrito com tempo de acesso muito rápido, evitando acessos mais demorados e maior complexidade da lógica para acessos às memórias externas. Pinos da FPGA são também muito importantes para a comunicação com outros dispositivos de processamento que eventualmente integrem o sistema completo [34].

Torres e al. [35] mencionam as vantagens do uso de FPGAs em aplicações de processamento de imagem e Kean et al. [36] comparam o desempenho de diferentes sistemas FPGA.

VanCourt e al. [37] relatam suas pesquisas para aceleração de processamento com resultados de um aumento no desempenho de até 1000 vezes, para o problema da análise de micromatrizes em DNA.

O artigo [38] exemplifica uma arquitetura de alto desempenho em FPGA, onde os autores implementaram um algoritmo de busca por vetores com tempos de execução de 8 a 340 vezes mais velozes do que a versão implementada em um

computador Pentium 4 de 3.5GHz, demonstrando toda a potencialidade da tecnologia FPGA para acelerar processamentos feitos em computadores convencionais.

Bonato [70] em sua tese de doutorado, implementa um sistema de processamento em paralelo para algoritmos de localização autômato em robótica, tendo inclusive operacionalizado um sistema físico utilizando o algoritmo SIFT no *hardware* desenvolvido. Autores em [4] desenvolveram sistemas com algoritmo de busca por padrão de imagem SIFT em FPGAs para aplicações em veículos autômatos para exploração de terreno em outros planetas.

O custo unitário de um dispositivo FPGA está normalmente muito abaixo do custo do processador em um PC, mas uma FPGA mostrou-se capaz de acelerar de 100 a 1000 vezes a aplicação serial executada em um computador. Apesar do apelo óbvio, as FPGAs são praticamente desconhecidas como aceleradores computacionais. O modelo de programação das FPGAs é estranho para vários desenvolvedores de *software*. Muitos irão dizer que o desenvolvimento de sistemas baseados em FPGA são desenvolvimentos de circuitos e não programação. Diversas habilidades são necessárias no desenvolvimento de circuitos em *hardware* eficientes. Estruturas computacionais eficazes não são necessariamente óbvias e nenhum compilador foi capaz de suplantar tais habilidades [39].

Implementações em *hardware* de algoritmos de processamento de imagem são muito difíceis devido à falta de flexibilidade para esta abordagem [48]. O uso de FPGAs como solução de sistemas de processamento de imagem envolve a diagramação, projeto, infra-estrutura de simulação e validação, desenvolvimento dos códigos em linguagem de descrição de *hardware*, técnicas de estruturação do sistema embarcado e de compilação adequadas. Este conjunto de etapas pode se tornar bastante intrincado, conforme a complexidade do próprio algoritmo a ser “paralelizado” para *hardware*.

A fim de solucionar a falta de flexibilidade e evitar o baixo nível para o desenvolvimento de sistemas de *hardware* como “aceleradores” de processos já criados para computador, o grupo de pesquisa em [39] propõe o desenvolvimento de ferramentas geradoras de código VHDL com interfaces em alto nível com os programadores. A ferramenta é baseada em uma linguagem projetada pelo grupo de pesquisa, chamada de *Logic Model Parameterization Markup Language* (LMPML).

Os autores em [40] desenvolveram um sistema de comparação e reconhecimento de íris para sistemas de segurança. Neste trabalho, os autores compararam o desempenho do algoritmo projetado executado em um PC Intel Xeon X5355 com o sistema em *hardware* executado no kit de desenvolvimento DE2 [41] com uma FPGA da família Cyclone II (EP2C35) da Altera com frequência de operação de 50Mhz. O tempo gasto por casamento de imagem no computador foi de 383ns enquanto que na FPGA o tempo consumido foi de 20ns, representando um ganho de 19 vezes. Os autores estimam também o tempo para execução em uma FPGA de estado-da-arte, a Stratix IV [73] com frequência de operação de 500MHZ. O ganho em desempenho seria de 192 vezes.

O trabalho em [42] propõe sistemas de reconhecimento visual de ações humanas desenvolvidos com uma FPGA Spartan-3 [43] da Xilinx. Já o trabalho em [44] apresenta um sistema de visão estéreo em FPGA que analisa as imagens através de duas câmeras e extraem o deslocamento de objetos das duas imagens. O deslocamento é contado em pixels e chamado de disparidade. Todas as disparidades formam um mapa de disparidades, o qual é o resultado do algoritmo de visão estéreo e permite o cálculo da distância dos objetos usando triangulação.

Os trabalhos em [51] e [45] apresentam sistemas de *hardware* implementados para a detecção de padrões em imagens em 3D.

A empresa VisionST [46] é uma empresa sul-coreana que desenvolve sistemas digitais e oferece produtos de visão computacional com combinação e detecção de objetos em tempo real com FPGAs. São sistemas bastante compactos e envolvem a utilização de câmeras e *softwares* em computadores.

Outro ponto importante em relação às FPGAs é que diferentemente de *workstations* de propósito geral, quando se esgotam os recursos do dispositivo-alvo não há nada a ser feito. No caso de projetos de sistemas multi-FPGAs é possível crescer a disponibilidade de recursos. O projetista entretanto deve desenvolver o sistema de interface e particionamento do algoritmo, além da transferência de dados e resultados entre os diferentes dispositivos FPGA. Até o momento, este tipo de solução normalmente requer uma maior compreensão dos sistemas lógicos digitais do que as ferramentas de projeto em alto nível parecem requerer [47].

Como nesta dissertação, no trabalho em [48] aos autores também partem de uma implementação em *software* já funcional para buscar soluções de maior desempenho com processamento em paralelo. No mencionado trabalho, o maior

consumo de tempo é sobre os cálculos de FFT 2D e transformadas polares. O objetivo da implementação era o de finalizar a busca em cada quadro em um tempo menor do que 33ms. O *hardware* implementado operava em uma frequência máxima de 60Mhz e o sistema eletrônico permitia o acesso a 4 memórias SRAM em paralelo, resultando em um sistema capaz de detectar em 27ms a posição e orientação do padrão buscado na imagem. A programação dos módulos foi feita utilizando a linguagem Handel-C. O sistema implementado também está em *pipeline* de modo que um resultado é obtido a cada ciclo de relógio. A resolução da imagem de entrada utilizada foi de 256x256 pixels.

Os fabricantes de FPGA incluem a Achronix Semiconductor, Actel, Altera, AMI Semiconductor, Atmel, Cypress Semiconductor, Lattice Semiconductor, QuickLogic, e Xilinx.

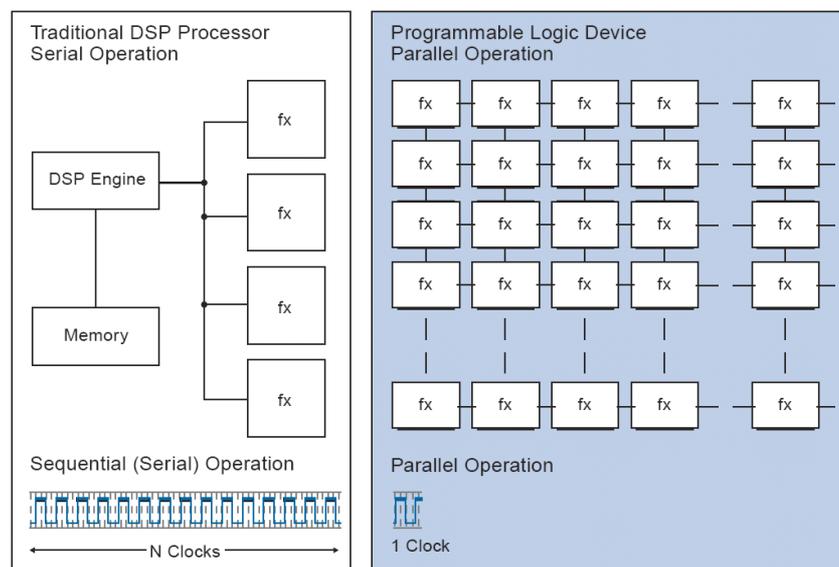
Maiores informações e detalhes comparativos das principais características dos dispositivos com as tecnologias mencionadas nos últimos tópicos pode ser encontrado em [7], onde os autores apresentam diversas abordagens de sistemas de alto desempenho para a solução de problemas de filtros de combinação.

### 3.3. PROCESSAMENTOS CONVENCIONAIS E EM PARALELO

As FPGAs oferecem a oportunidade de se projetar um *hardware* que está otimamente adaptado para um problema em especial. Assim, a otimização de um dado algoritmo para FPGA possui muito mais graus de liberdade do que em um microprocessador. Algumas destas possibilidades são [14]:

- Múltiplas unidades de processamento. Para cada tarefa pode-se decidir quantas unidades de processamento são executadas em paralelo e qual topologia em *pipeline* é mais adequada;
- Variabilidade do comprimento em bits dos sinais. Em um microprocessador padrão, os tipos de dados são fixados em múltiplos de bytes. As unidades de processamento possuem 32 bits ou mais de largura dos sinais necessários em aplicações com sensor de imagens, por exemplo. Este desperdício de recursos pode ser facilmente evitado em FPGAs, permitindo balancear desempenho e precisão;
- Unidades de processamento especiais. Um microprocessador convencional conhece apenas uma pequena quantidade de tipos de unidades de processamento. Para o processamento de tipos inteiros, é preciso incluir uma unidade lógica aritmética, um deslocador e uma unidade de multiplicação. Com FPGAs, qualquer tipo de unidade de processamento especial pode ser requerido para uma tarefa especial. Tais unidades de processamento dedicadas possuem maior desempenho e/ou consomem menos recursos que elementos de processamento padrão;
- Processamento das unidades *look-up tables*. Enquanto instruções multimídia aumentaram consideravelmente o desempenho de microprocessadores padrão, o cálculo de histogramas ou outras operações com *lookup tables* não podem ser aceleradas por este método. Os processadores FPGAs não apresentam este empecilho. É muito fácil implementar operações com *lookup tables*. Elas podem ser usadas para a realização de operações tais como multiplicação por uma constante, o cálculo de potenciação, raiz quadrada ou outra função não-linear.

O artigo em [49], escrito por fabricantes de FPGAs, compara os desempenhos das tecnologias de processamento paralelo com o de processadores usuais. Afirma-se que, apesar de os DSPs possuírem frequências de relógio muito mais elevadas que os FPGAs, estes últimos podem ser não só mais econômicos em termos de consumo de energia, como também podem executar um conjunto de tarefas matemáticas em um dado sistema, em tempo muito menor. Isto se deve à capacidade dos FPGAs de executarem centenas ou milhares de tarefas em um único ciclo de relógio (figura 18).



**figura 18** Diagrama de comparação entre o processamento convencional e o processamento paralelo de uma FPGA [50].

Um ponto ainda muito relevante em relação aos FPGAs é a flexibilidade destes dispositivos no que diz respeito à elevada quantidade de portas de entrada/saída para barramentos de comunicação, mesmo entre módulos projetados internamente [50].

### 3.4. FPGAs COMO CO-PROCESSADORES

A escolha ótima, segundo autores, para sistemas de visão embarcados está na combinação de um único FPGA com recursos suficientes de pinos de E/S para as interfaces, e uma interface de 64 bits com DPSs [34].

Graças a sua alta velocidade, grandes dispositivos FPGA executam suas aplicações independentemente do computador hospedeiro que fornece apenas funções de suporte. Sistemas típicos de processamento de imagem, entretanto, precisam normalmente de uma integração entre rotinas de aquisição de dados, processamento de imagem e dispositivo de disponibilização de imagens. Isto faz com que uma certa sub-classe de processadores FPGA, os co-processadores FPGA, se tornem ótimos candidatos para processamento de imagem.

Um co-processador FPGA está intimamente ligado a um computador convencional e otimizado para uma comunicação rápida com a CPU. O funcionamento é similar aos co-processadores mais antigos, mas executam, em contraste, algoritmos completos ao invés de operações únicas. Os co-processadores FPGA possuem normalmente apenas um ou poucos FPGAs em placas relativamente pequenas.

Em [51] é relatada a implementação de um sistema de alto desempenho tendo FPGAs como co-processadores, capaz de acelerar o casamento de padrões volumétricos de 100 a 1000 vezes, comparado com o desempenho em PCs.

A arquitetura geral com FPGA como co-processador é constituída dos seguintes componentes [14]:

- A FPGA é o *kernel* computacional onde os algoritmos são executados;
- O sistema de memórias é usado como um “pulmão” temporário ou para *lookup tables*;
- A interface de E/S é requerida para comunicação entre o sistema hospedeiro e o co-processador FPGA. *Drivers* específicos podem ser necessários.

O desempenho do co-processador FPGA depende justamente do desempenho dos três subsistemas citados.

Co-processadores FPGA introduzem um novo desafio em termos de programação FPGA devido à ligação estreita com a CPU. Há muita interação entre CPU e o co-processador FPGA, tratando-se de fato, de um co-projeto *software/hardware* onde a aplicação é dividida com uma parte sendo executada na FPGA e outra no microprocessador. A fim de simular corretamente o comportamento do sistema, é necessário o uso de um co-simulador. Como normalmente esta característica não é suportada por ferramentas HDL, uma linguagem de programação específica chamada de CHDL (C++ baseado em linguagem de descrição de *hardware*) foi criada [14].

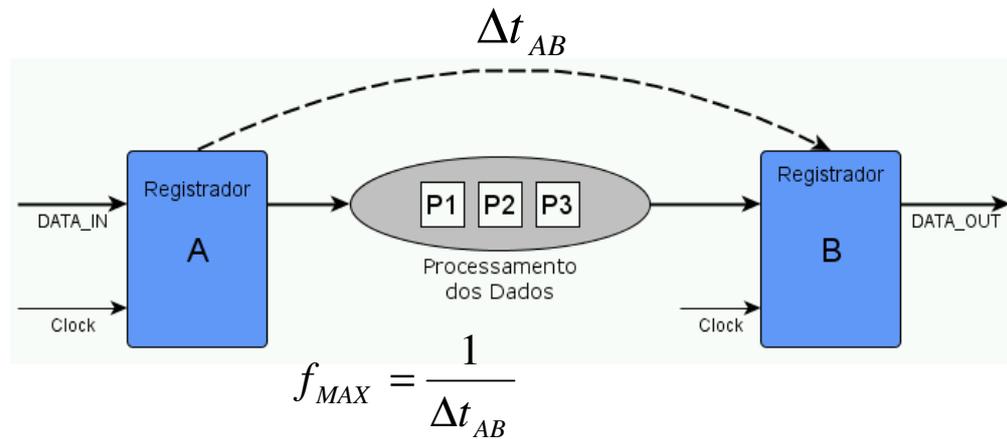
## 3.5. PROGRAMAÇÃO PARA HARDWARES DE ALTO DESEMPENHO

Como indicado anteriormente, o objetivo da primeira fase deste trabalho foi a análise das alternativas que viabilizassem atingir o maior desempenho possível do sistema de *hardware* (maior frequência de operação). Foram feitos testes e estudos que comprovaram a importância em manter todo o sistema sincronizado e garantir que todos os cálculos matemáticos operassem em *pipeline*, mesmo que isto implicasse em um aumento da latência e elevasse o tamanho em unidades lógicas dos módulos. Esta técnica permitiu atingir frequências mais elevadas e ainda garantir que, depois de alcançada a latência de cada módulo, qualquer resultado fosse disponibilizado em apenas um ciclo de relógio.

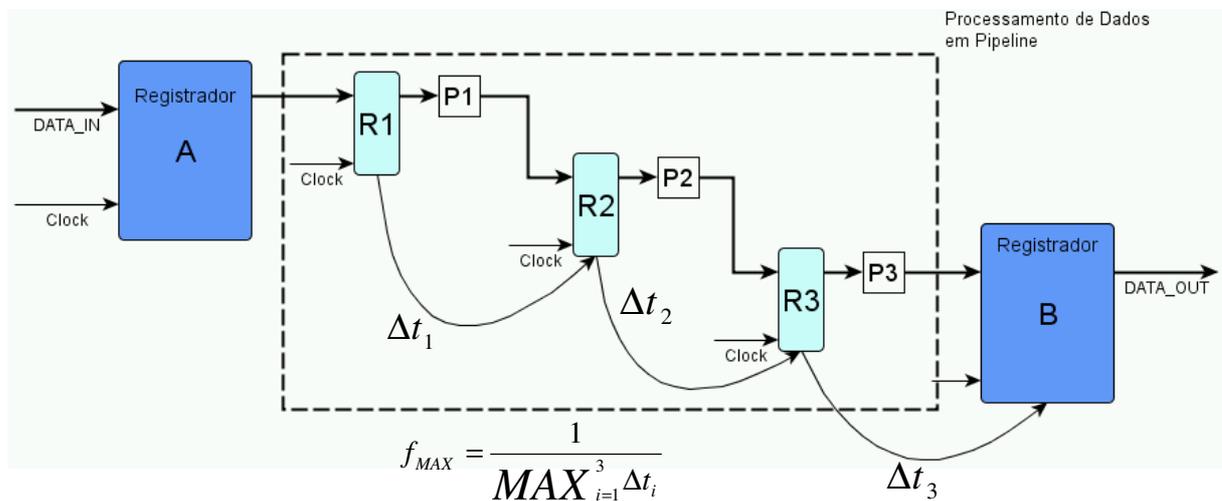
### 3.5.1. PIPELINE

Um projeto com *pipelines* conceitualmente funciona de modo similar a uma linha de montagem [52] em que a fila de material ou entrada de dados entra na linha de produção, passa por vários estágios de manipulação ou processamentos e deixa a linha como um produto acabado. Diferentes etapas na linha de montagem podem ser mais rápidas do que outras (diferentes latências em *hardware*). Assim, mesmo que certa etapa da linha de montagem seja mais lenta do que as demais, no momento em que o primeiro produto deixa a linha de montagem, os demais produtos na linha, estarão prontos a cada unidade de tempo ou ciclo de relógio.

Em projetos de alto desempenho é dada maior prioridade para a taxa de fluxo de dados do que para o tempo que algum módulo específico requer para propagar os dados pelo sistema (latência) [52]. Uma vantagem deste tipo de abordagem em *pipeline* é que um novo dado pode começar a ser processado antes que o dado anterior tenha acabado de ser processado. Projetos em *pipelines* são usados em praticamente todos os projetos de alto desempenho e a variedade de arquiteturas é ilimitada [52].



**figura 19** Diagrama para o processamento de dados sem o uso de *pipelines*.



**figura 20** Diagrama para o processamento de dados com o uso de três estágios de *pipelines*.

As figuras 19 e 20 comparam a frequência máxima que dois diferentes sistemas de processamento são capazes de atingir, sem o uso de estruturas em *pipeline* e com tais estruturas. A frequência máxima é tida como a soma dos tempos de propagação entre as etapas intermediárias. Com o uso de registradores intermediários em *pipeline* a frequência máxima de operação é dada pelo maior tempo de propagação no processamento entre os registradores.

### 3.5.2. OS FPGAS "ACHRONIX"

A empresa Achronix disponibilizou recentemente as mais rápidas FPGAs do mercado capazes de operar com frequências além de 1GHz [53]. O grande diferencial entre as FPGAs da Achronix e as demais é a estrutura de rede de registradores criada fisicamente no próprio dispositivo. O nome dado a essa estrutura é de *PicoPIPE* e faz com que todos os sinais que se intercomunicam dentro do dispositivo passem por sistemas físicos de *pipeline*, como ilustrado na figura 21.

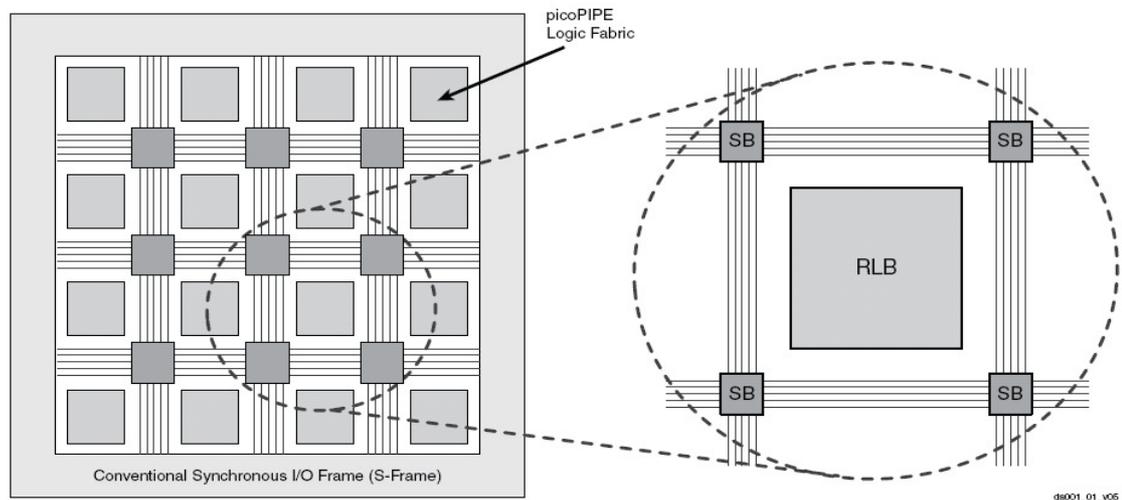


figura 21 Arquitetura das FPGAs Achronix [54].

As FPGAs da Achronix atingem maior desempenho em comparação às FPGAs convencionais devido aos estágios em *pipeline* para os quais toda a lógica combinatória é automaticamente roteada, não havendo necessidade de projetá-los especificamente em VHDL. Assim, como na figura 22, entre cada elemento de lógica há uma conexão física em silício, em *pipeline* que faz a ligação com a lógica seguinte [53].

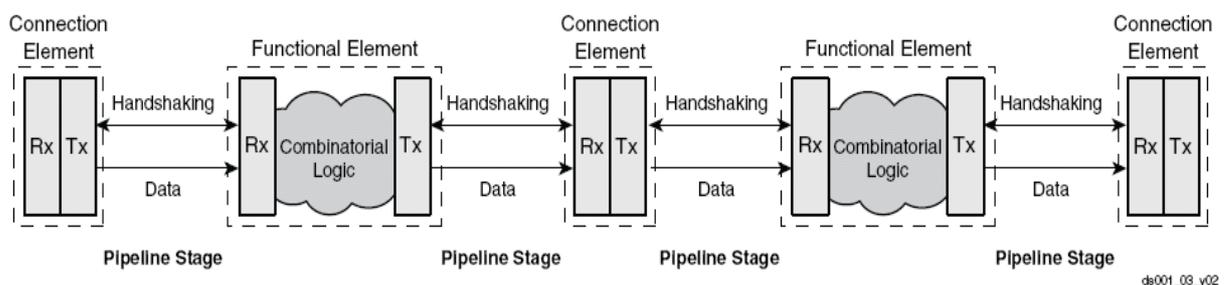


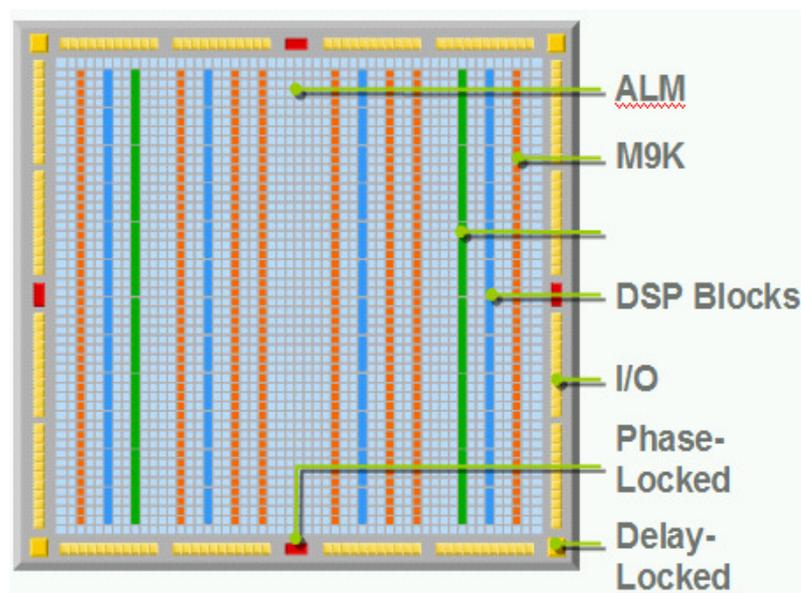
figura 22 Estrutura da arquitetura *PicoPIPE* desenvolvida pela empresa Achronix [54].

### 3.6. TECNOLOGIA DO DISPOSITIVO UTILIZADO

A fim de se evitar acessos às memórias externas, na leitura dos dados dos pixels da imagem *A*, optamos por trabalhar com uma FPGA capaz de armazenar internamente uma imagem completa em tons de cinza com resolução de 640x480. O dispositivo escolhido foi o da família Stratix III, cujo código é EP3SL340H1152C3 e possui memória interna de 16 Mbits. Alguns outros trabalhos (como por exemplo, em [54]) também propõem a utilização da memória interna da FPGA para processar a imagem analisada de modo a não se submeter aos tempos de acesso para interface com memórias externas.

Trata-se de um dispositivo com 338.000 elementos lógicos, em encapsulamento *Fine Line Ball Grid Array (FBGA)* de 1104 pinos. Será apresentado a seguir um breve resumo das características da família de FPGA Stratix III da Altera. Uma descrição mais detalhada dos dispositivos pode ser encontrada em [55].

A figura 23 apresenta o diagrama da arquitetura da família Stratix III. Esta arquitetura suporta um bom desempenho no processamento de sinal digital pois possui o dobro de blocos DSP comparado a outros tipos de dispositivos convencionais e/ou antecessores. Também apresenta mais blocos multiplicadores/acumuladores, além de ter o mais alto desempenho e baixo custo [56].



**figura 23** Diagrama com os principais recursos das FPGAs da família Stratix III [56].

Os dispositivos da família Stratix III da Altera têm as seguintes características comuns:

- Implementados em tecnologia 65nm;
- 48.000 a 338.000 elementos lógicos;
- 2.430 a 20.497 Kbits de memória tipo *TriMatrix* [55];
- Blocos DSP de alto desempenho que oferecem implementação dedicada de multiplicadores 9x9, 12x12, 18x18 e 36x36 (até 550 MHz), funções multiplica-accumula e filtros FIR;
- Até 16 relógios globais, 88 relógios regionais, e 116 relógios periféricos por dispositivo;
- Até 12 PLLs (*Phase-Locked Loops*) por dispositivo;
- Até 1.104 pinos de E/S diagramados em 24 bancos modulares.

A figura 24 apresenta os modelos de dispositivos Stratix III e as principais características físicas presentes para cada um destes. A linha indicada pela seta indica o modelo selecionado para este trabalho.

	Device/ Feature	ALMs	LEs	M9K Blocks	M144K Blocks	MLAB Blocks	Total Embedded RAM Kbits	MLAB RAM Kbits(2)	Total RAM Kbits(3)	18x18-bit Multipliers (FIR Mode)	PLLs (4)
Stratix III Logic Family	EP3SL50	19K	47.5K	108	6	950	1,836	297	2,133	216	4
	EP3SL70	27K	67.5K	150	6	1,350	2,214	422	2,636	288	4
	EP3SL110	43K	107.5K	275	12	2,150	4,203	672	4,875	288	8
	EP3SL150	57K	142.5K	355	16	2,850	5,499	891	6,390	384	8
	EP3SL200	80K	200K	468	36	4,000	9,396	1,250	10,646	576	12
	EP3SE260	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12
	EP3SL340	135K	337.5K	1,040	48	6,750	16,272	2,109	18,381	576	12
Stratix III Enhanced Family	EP3SE50	19K	47.5K	400	12	950	5,328	297	5,625	384	4
	EP3SE80	32K	80K	495	12	1,600	6,183	500	6,683	672	8
	EP3SE110	43K	107.5K	639	16	2,150	8,055	672	8,727	896	8
	EP3SE260 (1)	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12

figura 24 Características dos modelos de FPGAs da família Stratix III [56].

### 3.6.1. MÓDULOS DE LÓGICA ADAPTATIVA

A base de blocos lógicos da Stratix III corresponde aos módulos ALM (*Adaptive Logic Module*) os quais contêm uma variedade de recursos baseados em *LookupTables* e podem ser divididas em duas LUT adaptativas combinacionais (ALUT) e dois registradores. Com até 8 entradas para os dois ALUT combinacionais, uma ALM pode implementar várias combinações das duas funções (ffigura 25).

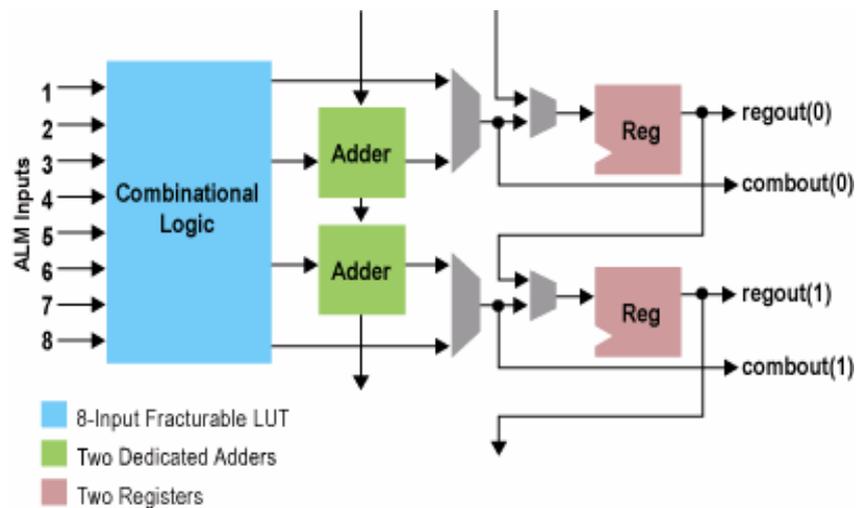


figura 25 Diagrama em alto nível de uma unidade ALM para a Stratix III [56].

A figura 26 apresenta exemplos para os módulos de *lookup tables* na seleção da função lógica a ser implementada e os diferentes modos de instanciação.

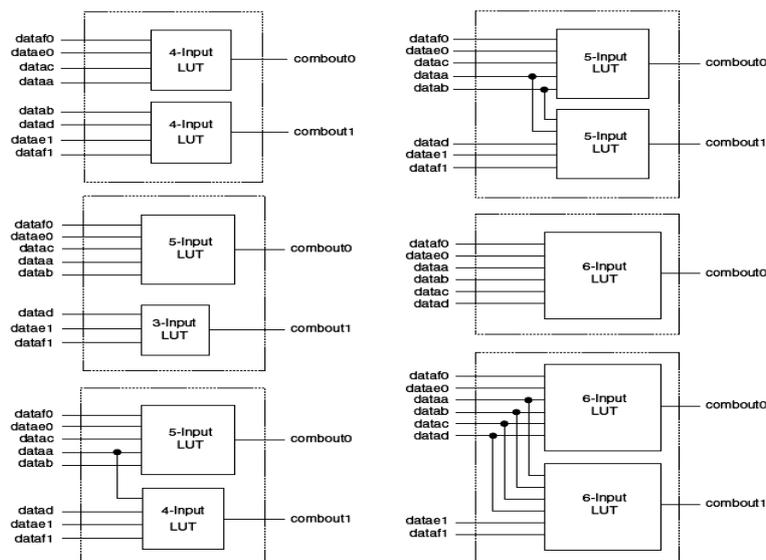


figura 26 Diagramas com os modos das entradas para as *Lookup Tables* [56].

### 3.6.2. BLOCOS DE MATRIZ LÓGICA

Cada LAB (*Logic Array Block*) consiste de dez ALMs. As unidades LAB da Stratix III possuem um novo derivativo chamado de MLAB (*Memory LAB*) o qual adiciona memória SRAM baseada em LUT para o LAB. São muitas as opções de configuração destes blocos SRAM de memórias.

A figura 27 apresenta a estrutura de um módulo LAB. Nela é possível identificar a arquitetura de interconexões entre os módulos que possuem diferentes comprimentos e desempenhos.

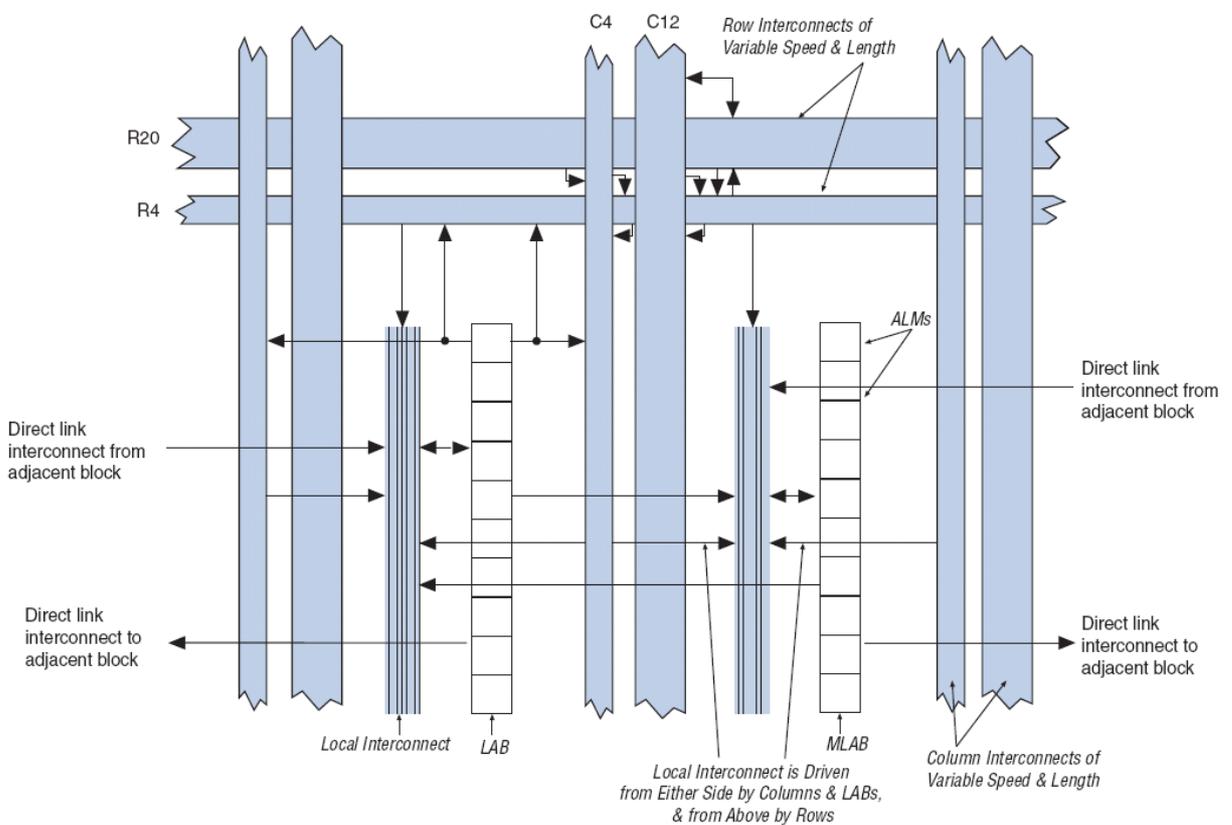


figura 27 Estrutura do módulo LAB para a família Stratix III [56].

### 3.6.3. MEMÓRIAS INTERNAS

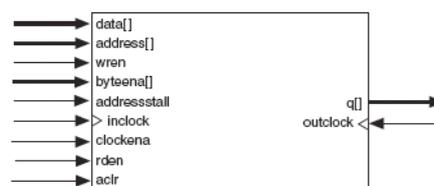
#### 3.6.3.1. BLOCOS DE MEMÓRIA *TRIMATRIX*

Os blocos de memória nos dispositivos Stratix III passam a ter maiores possibilidades de configuração e desempenho em comparação com os dispositivos das gerações anteriores. Estes módulos de memória foram chamados de *TriMatrix*. A figura 28 apresenta uma relação das configurações e modos de operação, separando-os inclusive pelo desempenho máximo possível para cada um.

Feature	MLABs	M9K Blocks	M144K Blocks
Maximum performance	600 MHz	580 MHz	580 MHz
Total memory bits (including parity bits)	640 (in ROM mode) or 320 (in other modes)	9,216	147,456
Configurations (depth × width) (1)	16 × 8	8 K × 1	16 K × 8
	16 × 9	4 K × 2	16 K × 9
	16 × 10	2 K × 4	8 K × 16
	16 × 16	1 K × 8	8 K × 18
	16 × 18	1 K × 9	4 K × 32
	16 × 20	512 × 16	4 K × 36
		512 × 18	2 K × 64
		256 × 32	2 K × 72
	256 × 36		
Parity bits	✓	✓	✓

**figura 28** Sumário das características para os blocos de memória *TriMatrix* [56].

As figura 29, 30 e 31 apresentam os diagramas para os modos de configuração existentes para as memórias internas. A imagem A, com resolução de 640x480 é armazenada nestas memórias internas, onde é feita a busca pela imagem de máscara. O tempo de acesso às memórias internas da FPGA é menor do que o acesso às memórias externas, como em uma eventual placa eletrônica.



**figura 29** Memória tipo *Single-Port* [32].

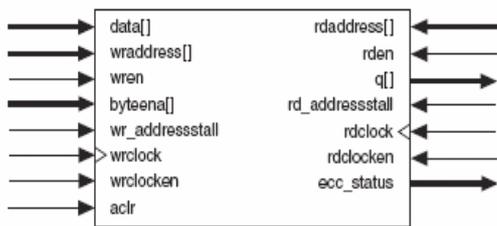


figura 30 Memória tipo *Simple Dual-Port* [32].

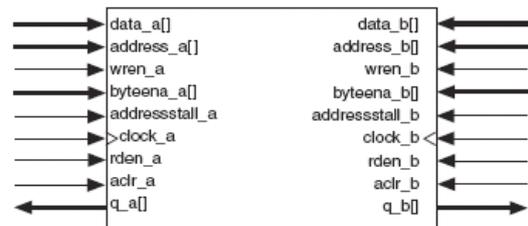


figura 31 Memória tipo *True Dual-Port* [32].

### 3.6.3.2. MODO DESLOCADOR DE REGISTRADOR

Todos os blocos de memórias da Stratix III suportam o modo deslocador de registrador, os quais se mostram adequados para aplicações de processamento digital de sinais, tais como filtros FIR, geração de números pseudo-aleatórios, filtros multi-canais e funções de correlação. Estas e outras aplicações de processamento requerem o armazenamento local de dados, tradicionalmente implementado em *flip-flops* convencionais que consomem muitas células lógicas para grandes deslocadores de registrador. Uma alternativa mais eficiente é utilizar memórias embarcadas economizando células lógicas e recursos de roteamento. É possível cascatear blocos de memória para implementar maiores deslocador de registrador (figura 32) [32].

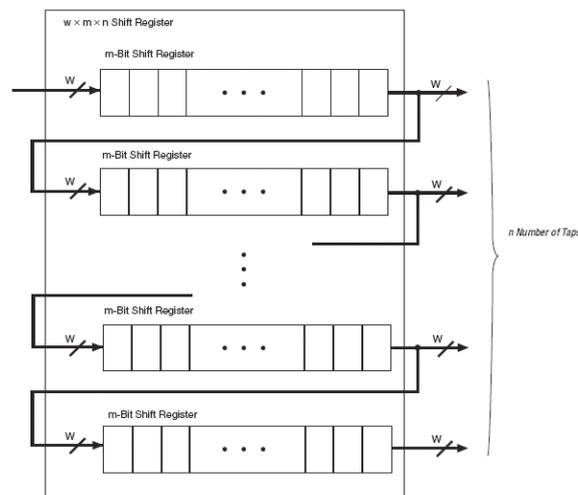


figura 32 Implementação em cascata de *Shift Registers* em blocos de memória [32].

### 3.6.3.3. DSPs INTERNOS

A família de dispositivos Stratix III possui blocos otimizados de DSP de alto desempenho e consistem de uma combinação de elementos dedicados que realizam multiplicação, adição, subtração, acumulação-soma e operações dinâmicas de deslocamentos. Em conjunto com o barramento de comunicação e estruturas *TriMatrix* de memórias, é possível configurá-los para a realização de funções sofisticadas em ponto-fixe e ponto flutuante.

Cada dispositivo possui de 2 a 7 colunas de blocos DSP que implementam eficientemente funções de multiplicação, multiplicação-adição, multiplicação-acumula (MAC) e deslocamento dinâmico. A figura 33 apresenta os modos de operação para os blocos DSP em combinação com os recursos disponíveis.

Mode	Multiplier in Width	# of Mults	# per Block	Signed or Unsigned	RND, SAT	In Shift Register	Chainout Adder	1st Stage Add/Sub	2nd Stage Add/Acc
Independent Multiplier	9-bits	1	8	Both	No	No	No	—	—
	12-bits	1	6	Both	No	No	No	—	—
	18-bits	1	4	Both	Yes	Yes	No	—	—
	36-bits	1	2	Both	No	No	No	—	—
	Double	1	2	Both	No	No	No	—	—
Two-Multiplier Adder <sup>(1)</sup>	18-bits	2	4	Signed <sup>(4)</sup>	Yes	No	No	Both	N/A
Four-Multiplier Adder	18-bits	4	2	Both	Yes	Yes	Yes	Both	Add Only
High Precision Multiplier Adder	18 × 36-bits	2	2	Both	No	No	No	—	Add Only
Multiply Accumulate	18-bits	4	2	Both	Yes	Yes	Yes	Both	Both
Shift <sup>(2)</sup>	36-bits <sup>(3)</sup>	1	2	Both	No	No	—	—	—

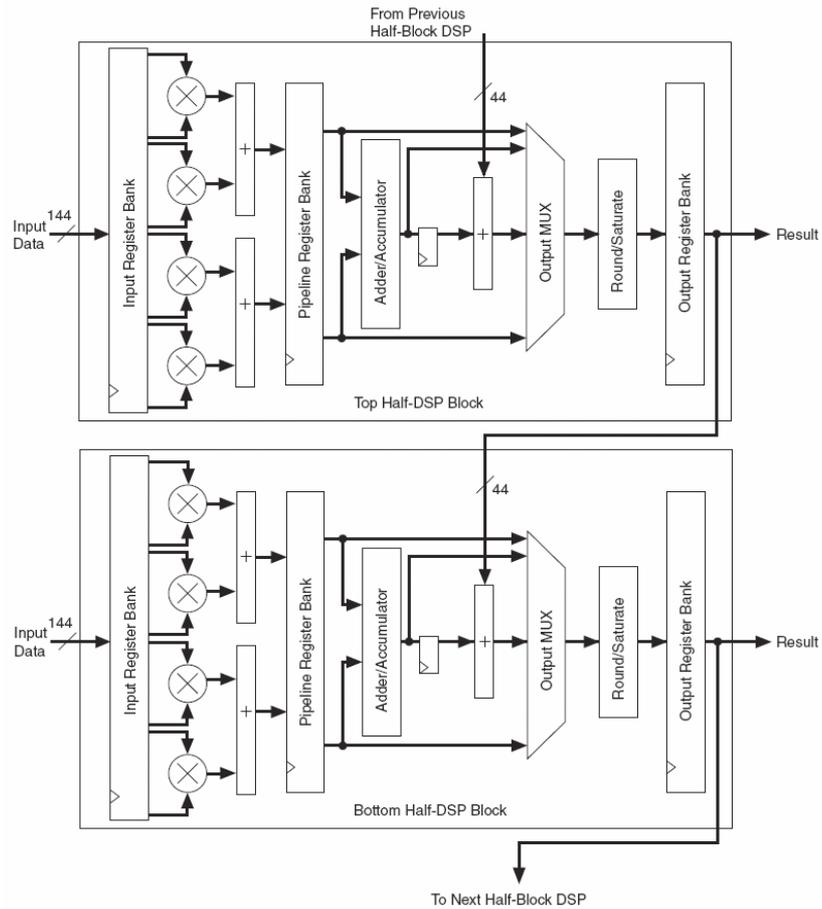
figura 33 Modos de operação dos blocos DSP na Stratix III [33].

Os pontos mais importantes da arquitetura DSP para a Stratix III são [53]:

- Operandos de multiplicação de alto desempenho, otimizados em nível de consumo, totalmente registrados e com *pipeline*;
- Largura de dados de 9bits, 12bits, 18bits e 36bits;
- Multiplicações complexas com 18bits;
- Eficiência em formatos para ponto-flutuante (24 bits para precisão única e 53 bits para precisão dupla);
- Entradas do tipo com sinal e sem sinal;

- Modos em cascata para propagação do resultado ao bloco DSP seguinte sem o auxílio de lógica externa.

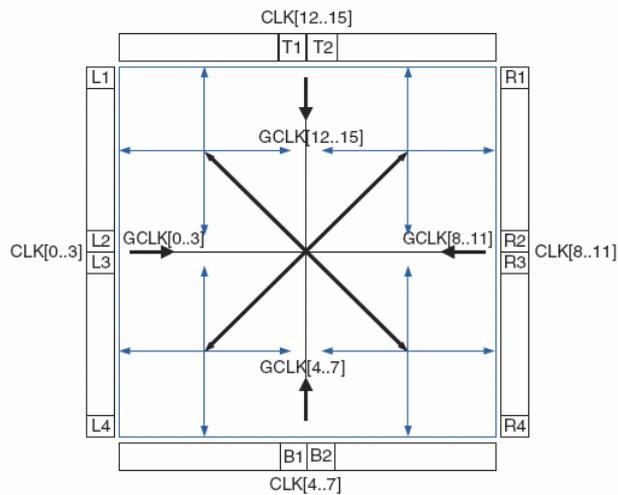
A figura 34 demonstra a capacidade de configuração dos blocos DSP para implementação de funções complexas de multiplicação-soma-e-acumula, inclusive com cascadeamento do resultado para blocos seguintes.



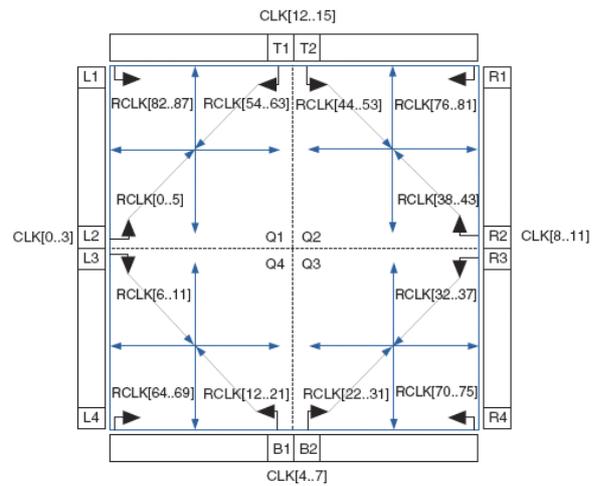
**figura 34** Quatro blocos DSP multiplica-soma-e-acumula [33].

### 3.6.3.4. REDES DE RELÓGIO

A arquitetura de redes e interconexões dos sinais de relógio é um fator muito importante para o desempenho do *hardware* como um todo. Na Stratix III há uma rede completa de relógios globais (figura 35) separados por bancos específicos, nos quais são encontradas outras estruturas de interconexão de sinais de relógio que são os relógios regionais (figura 36).



**figura 35** Diagrama de redes de relógio globais [33].



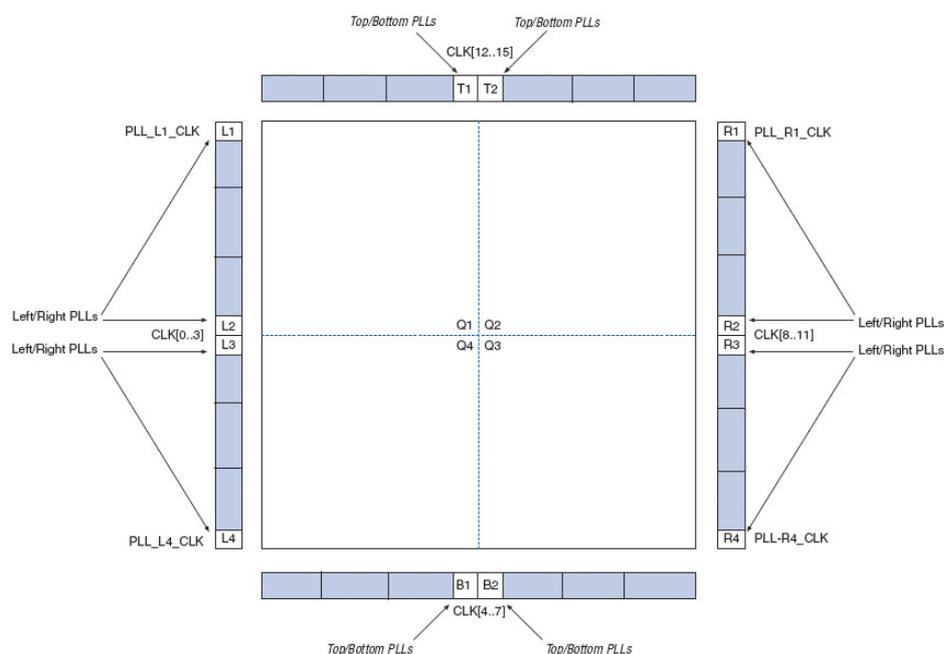
**figura 36** Redes de relógio regionais [33].

### 3.6.3.5. PLLs

O PLL (*Phase Locked Loop*) é o método mais comum de se produzir oscilações estáveis de altas frequências em equipamentos modernos de comunicação. Técnicas de PLL surgiram na década de 1930 [57].

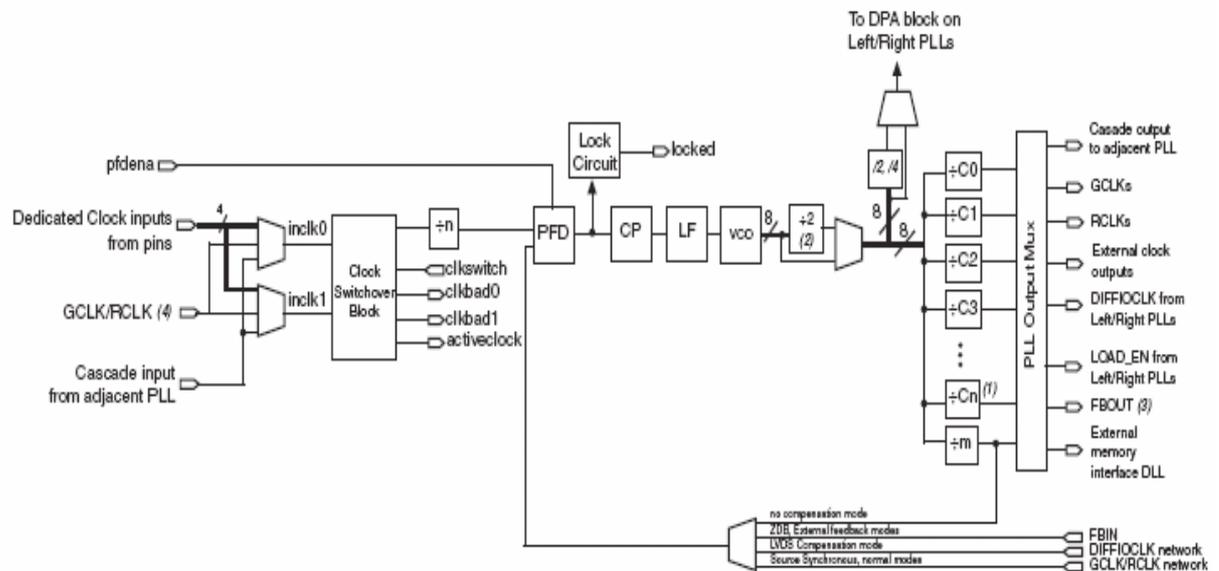
Os circuitos PLL nas FPGAs são utilizados para gerar frequências de relógio bastante complexas, permitindo o ajuste no valor destas frequências (gerando frequências muitas vezes superiores às de entrada, presentes na própria placa eletrônica) e na fase. Estes módulos são imprescindíveis em sistemas digitais, no acesso às diferentes memórias e na adequação do sistema ao desempenho conseguido na etapa de roteamento do *hardware* para a FPGA selecionada.

Os dispositivos Stratix III possuem até 12 PLLs que oferecem o gerenciamento robusto dos relógios para uso em relógios externos na síntese do circuito interno e no gerenciamento de interfaces rápidas de E/S. A figura 37 aponta a localização dos PLLs no dispositivo. Existem PLLs em dois planos diferentes, superior e inferior, distribuídos conforme indicado na figura.



**figura 37** Localização dos PLLs na Stratix III [33].

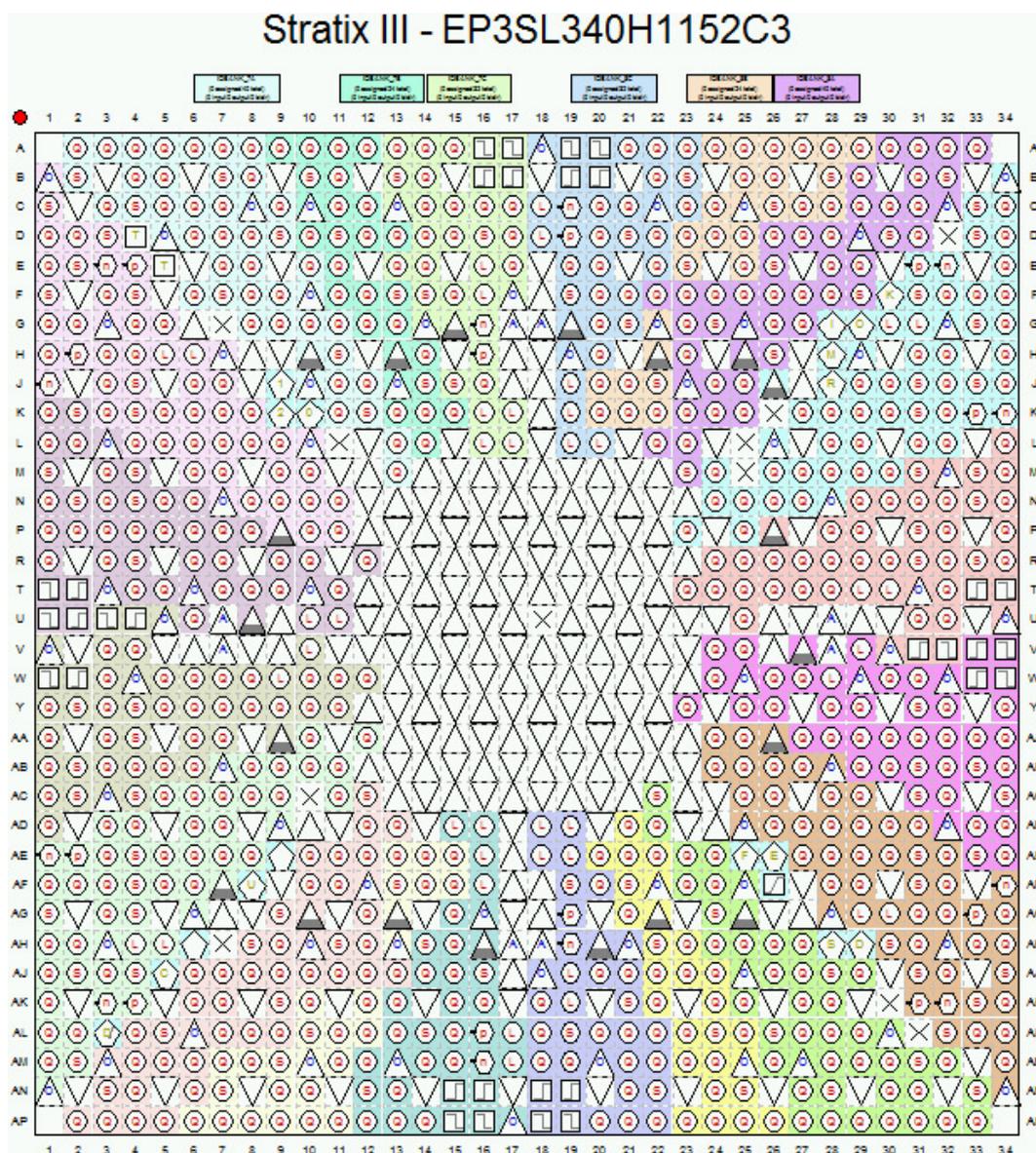
A figura 38 demonstra o diagrama de blocos de um PLL da família Stratix III ilustrando a complexidade deste módulo. É possível verificar os módulos de controle das freqüências e da fase, registradores e demais circuitos.



**figura 38** Diagrama de bloco para os PLL da família Stratix III [33].

### 3.6.3.6. PINOS DE ENTRADA/SAÍDA

Os pinos de entrada e saída de uma FPGA são os responsáveis pela comunicação da FPGA com o mundo externo, seja no acesso aos periféricos de memória, monitores, relógios de entrada ou interfaces de comunicação com outros processadores. A figura 39 obtida através do programa Quartus II da Altera, demonstra os diversos bancos de E/S presentes na FPGA selecionada para este projeto e que possui um total de 1152 pinos.



**figura 39** Bancos de E/S para os dispositivos Stratix III.

Na figura 40 é apresentado um diagrama de blocos para os pinos de entrada/saída nas FPGAs da família Stratix III, ilustrando sua complexidade. Cada pino de E/S pode ser configurado como entrada, saída ou como bidirecional. Há alguns pinos na FPGA que são pinos especiais, para trocas de dados (com memórias ou barramentos) em alta velocidade; e pinos que são específicos para entrada de relógios externos.

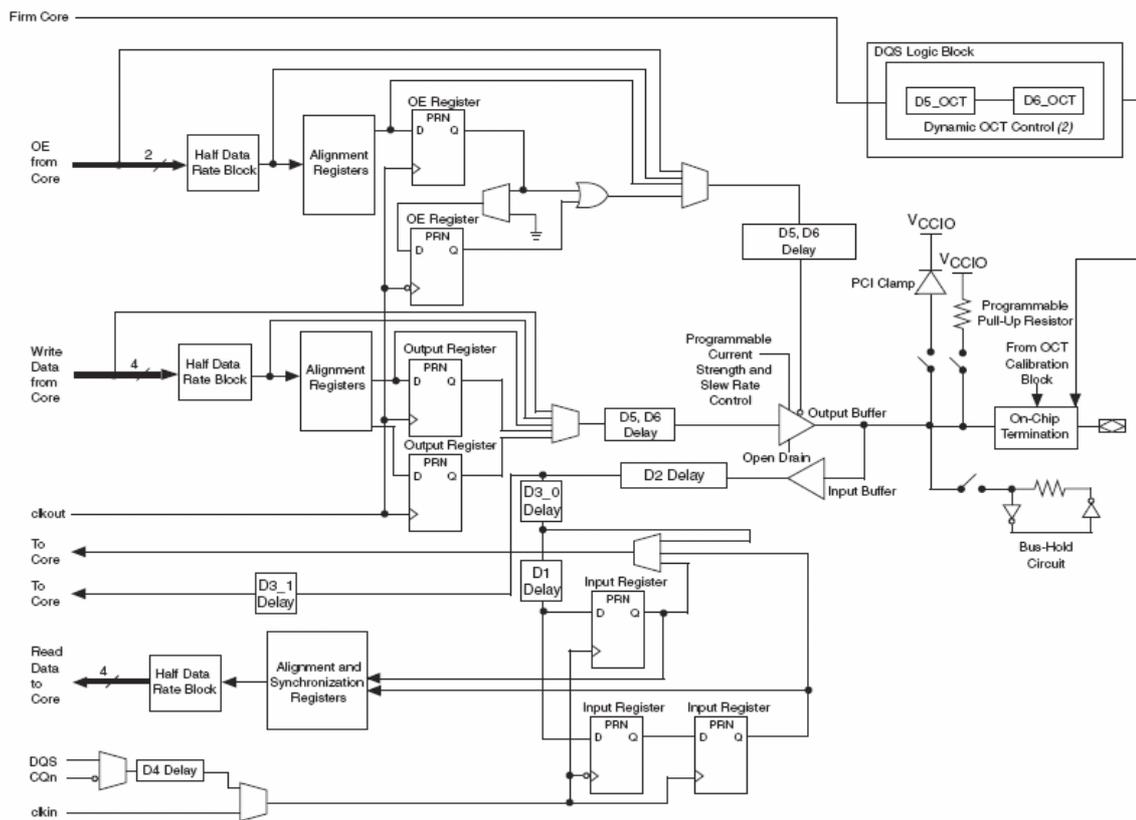


figura 40 Diagrama de um pino de E/S da Stratix III [33].

## 3.7. ETAPAS PARA PROJETOS EM FPGA

Para este trabalho, selecionamos os dispositivos da Altera, utilizando suas ferramentas de síntese, roteamento e análise de frequências. Para todas as simulações, foi utilizado o software ModelSim [58] da empresa MentorGraphics [59] que permite explorar o comportamento do processamento paralelo e analisar os resultados gerados pelo *hardware* em VHDL.

Nos próximos tópicos serão apresentadas as principais etapas envolvidas no projeto e execução de *hardware* em dispositivos FPGA.

### 3.7.1. ETAPA DE DESCRIÇÃO DE *HARDWARE*

O fluxo de projetos de uma FPGA se inicia pela especificação do circuito a se desenvolver. A especificação pode ser um algoritmo que será implementado, ou uma funcionalidade deste algoritmo ou um esquema elétrico a ser implementado. Uma especificação mal feita leva a um ciclo de desenvolvimento muito maior e eventualmente a circuitos que não funcionem na prática. Esta etapa define se o ciclo de projeto será maior ou menor e conseqüentemente, a qualidade do projeto desenvolvido [21].

A partir da especificação completa, o próximo passo é implementar o projeto criando os arquivos de descrição de *hardware* através de uma linguagem específica de descrição de *hardware* (VHDL, Verilog, ABEL, AHDL ou outra).

Quanto maior a complexidade da FPGA, mais importante passam a ser as ferramentas de programação confiáveis, rápidas e eficientes. O estado da arte das linguagens de descrição de *hardware* (HDL) são aquelas usadas há muito tempo no projeto de circuitos integrados. As mais comuns das linguagens são o VHDL e o Verilog [14].

No caso do projeto apresentado nesta dissertação, o circuito foi implementado utilizando-se a linguagem de descrição de *hardware* VHDL [ 60 , 61 ]. Na implementação do circuito especificado devem ser tomados os devidos cuidados e adotados critérios de boas práticas de projeto necessárias à implementação de sistemas digitais de alto desempenho.[62;63;64;65 apud 21].

Após o desenvolvimento dos arquivos em *VHDL* com a descrição do *hardware* a ser implementado, deve-se simular a funcionalmente do circuito para verificar se o circuito desenvolvido está funcionando conforme o especificado. Para isto, a descrição de *hardware* é submetida a um simulador para a verificação da correspondência entre a especificação e o código.

Um processo iterativo de simulações e detalhamento dos elementos da estrutura é executado até ser atingida uma descrição que permita a síntese, e até que as simulações assegurem a equivalência entre a especificação do projeto e a descrição proposta [66].

### **3.7.2. ETAPA DE SÍNTESE**

A segunda etapa de desenvolvimento é a etapa de síntese (chamada de etapa de *Analysis and Synthesis* no programa Quartus II) em que a ferramenta de síntese executa o processo de inferência e interligação das estruturas necessárias para o circuito a ser gerado a partir da descrição. Nesta etapa, é criado um circuito no nível RTL (*Register Transfer Level*), empregando primitivas disponíveis na ferramenta como comparadores, somadores, registradores, portas lógicas e também blocos de memórias internas e blocos DSP que possam eventualmente estar presentes no dispositivo FPGA selecionado para o projeto. O circuito gerado nessa etapa não está associado a nenhuma tecnologia de fabricação ou dispositivo em particular, e não está, necessariamente, otimizado [66].

### **3.7.3. ETAPA DE ROTEAMENTO**

Após a etapa de síntese do projeto, é feito o posicionamento dos elementos lógicos sintetizados no dispositivo selecionado e o roteamento (*Fitter e Place and Route* no programa Quartus II) dos sinais de interligação dos elementos lógicos. A diferença entre o circuito obtido na etapa anterior, de síntese, e aquele gerado no roteamento está relacionada aos elementos utilizados pois, enquanto no nível RTL são empregadas primitivas genéricas da ferramenta, no roteamento o circuito contém unicamente elementos disponíveis na tecnologia do dispositivo FPGA selecionado para o projeto.

Uma minimização é executada nessa etapa, na qual dois parâmetros geralmente conflitantes são considerados: otimização de custo (área do dispositivo) e otimização em termos de desempenho. Um dos principais resultados dessa fase é um arquivo contendo uma rede de ligações entre os elementos internos disponíveis na tecnologia ou circuito lógico integrado utilizado. Com isso, é possível incluir a re-síntese de partes do circuito para que se obtenha um melhor desempenho do circuito final ou para reduzir o tamanho do circuito identificando partes do circuito equivalentes que possam ser eliminadas [21].

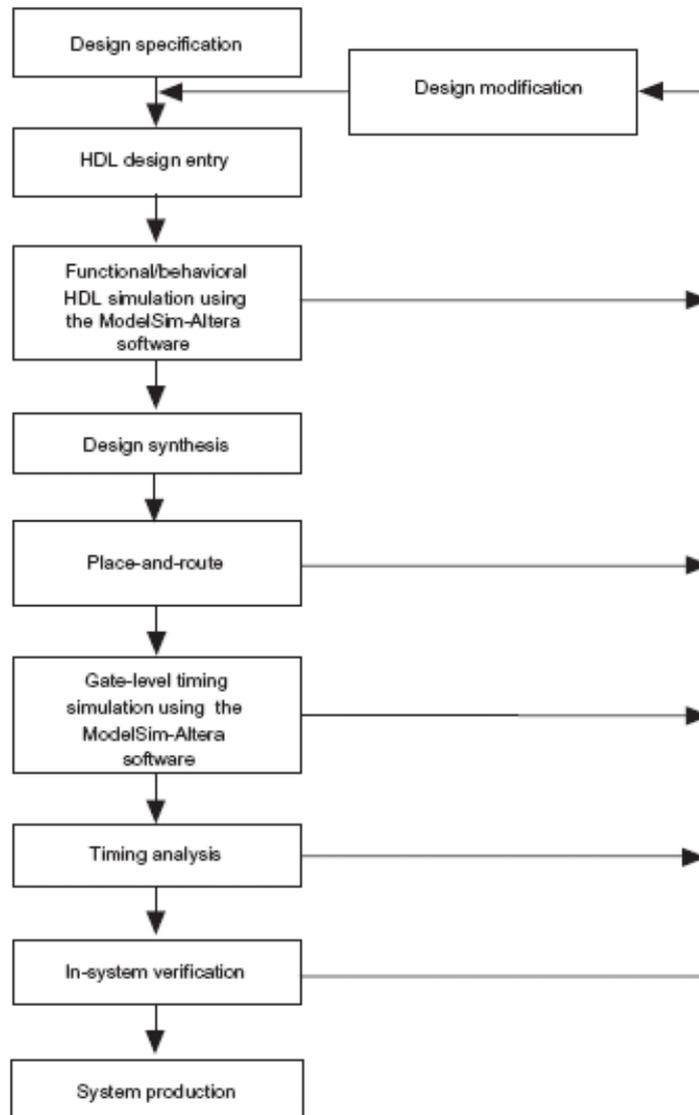
É nesta parte do desenvolvimento que a ferramenta de síntese, valendo-se de inúmeros parâmetros de robustez, realiza o processo de leiaute do projeto sintetizado para o modelo de circuito programável executado. Este processo envolve inúmeros cálculos e pode exigir até mesmo algumas horas de processamento do computador, dependendo do tamanho tanto do dispositivo programável escolhido quanto do próprio projeto sintetizado.

Após a conclusão do leiaute dos elementos lógicos no dispositivo, o *software* de desenvolvimento gera os arquivos de configuração (*Assembler* no programa Quartus II) que, ao serem carregados no dispositivo físico, permitem que o mesmo seja configurado com o circuito compilado. Este arquivo pode ser carregado através de uma interface JTAG diretamente na lógica do dispositivo, ou pode ser gravado em um dispositivo de configuração. Há ainda a possibilidade deste arquivo ser armazenado na memória não volátil de um processador e posteriormente transferido pelo processador para a FPGA [21].

Na seqüência, é criado um modelo dos tempos de propagação do circuito (*Gate Level Timing Analysis* no programa QuartusII), levando-se em conta os tempos de propagação internos do componente, o posicionamento dos elementos lógicos no dispositivo e o roteamento dos sinais entre os elementos lógicos. A partir deste modelo de tempos de propagação são calculadas as frequências de operação do circuito, bem como os tempos de propagação de entrada e saída.

Em uma última etapa do projeto este modelo de tempos de propagação é usado para simular novamente o circuito desenvolvido levando em conta, desta vez, os tempos de propagação internos e de entrada e saída do dispositivo em consideração. Esta simulação pode ser feita tanto pelo Quartus II [67] quanto por ferramentas de outros fabricantes, como o ModelSim [58].

Na figura 41 é apresentado o fluxo de projeto típico para dispositivos da Altera utilizando a ferramenta de desenvolvimento integrada da Altera, o Quartus II e o *software* de simulação ModelSim..



**figura 41** Fluxo de projeto da Altera integrando os softwares Quartus II e ModelSIM da MeteorGraphics [68].

Utilizamos para todos os desenvolvimentos desta dissertação de mestrado, a versão *WebEdition* gratuita disponível para no sítio da Altera [28]. A empresa disponibiliza para compra a versão completa do *software*. As diferenças entre as duas versões podem ser verificadas em [69] mas estão relacionadas com a maior disponibilidade de ferramentas de otimização e a possibilidade de gerar *hardwares* para gravação na versão completa do programa. A Altera também disponibiliza uma versão gratuita do *software* ModelSIM.

### **3.7.4. SIMULAÇÕES**

A etapa de simulação, verificação e validação é uma etapa fundamental no projeto de sistemas digitais. Esta etapa pode requerer mais tempo de projeto do que o despendido para os *hardwares* efetivos. Entretanto, uma etapa de simulação feita adequadamente, diminui enormemente as chances de que o sistema real não funcione corretamente ou não funcione de forma alguma.

Os códigos VHDL nesta etapa são códigos exclusivos para simulação e não integram o sistema final e muitas vezes, inclusive, são códigos não-sintetizáveis, ou seja, códigos que não podem ser convertidos em nenhum *hardware*.

#### **3.7.4.1. BIBLIOTECA “TEXTIO”**

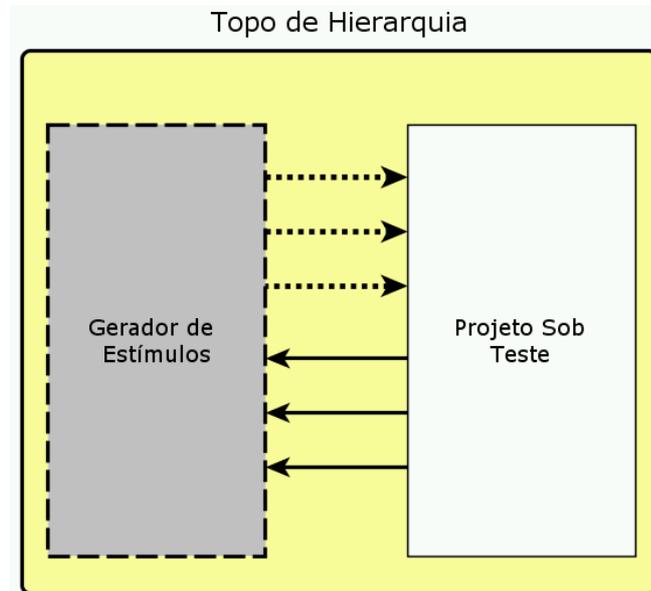
Existem muitas bibliotecas específicas para o uso em simulações e que contém funções ou tipos de variáveis para aplicações bem definidas. Neste trabalho foram projetados módulos VHDL específicos de manipulação de arquivos de texto os quais possuem bibliotecas específicas em VHDL. São as bibliotecas tipo TextIO (*Textual Input and Output*) [61] que contém tipos, processos e funções para viabilizar a escrita e leitura em arquivos no formato ASCII.

#### **3.7.4.2. TESTBENCHS**

Um simulador requer duas entradas: a descrição do projeto e os estímulos que o controlam. Alguns projetos podem ser auto-estimulados e não necessitam de nenhum estímulo externo, mas na maior parte das vezes, os projetistas utilizam um módulo VHDL de *testbench* para gerenciar a simulação.

A estrutura do projeto assemelha-se à figura 42. O módulo de topo de hierarquia instancia dois componentes: o primeiro é o DUT (*Design Under Test*) e o segundo é o controlador de estímulos. Estes componentes são conectados com sinais que representam o ambiente externo do DUT. O módulo topo de hierarquia

não contém nenhum sinal externo, apenas sinais internos que conectam os componentes instanciados [61].



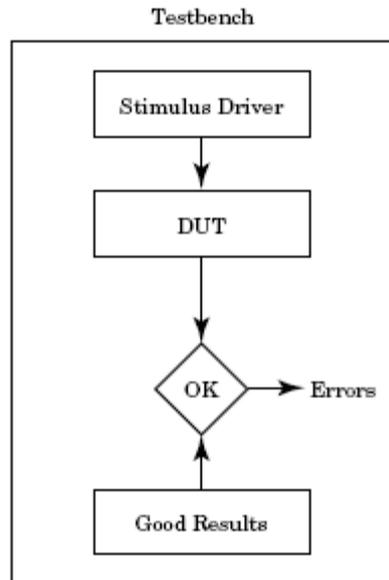
**figura 42** Estrutura de topo de hierarquia para simulação e testes [62].

Um módulo de *testbench* é usado para verificar a funcionalidade de um projeto em cada etapa da metodologia de projetos HDL. Quando o projetista faz uma pequena mudança para corrigir um erro, a modificação pode ser testada para garantir que esta não afeta as demais partes do projeto. Novas versões do projeto podem ser analisadas conforme os resultados esperados para verificações de compatibilidade.

O módulo *testbench* é o arquivo topo de hierarquia do projeto e instancia o DUT, provendo os estímulos necessários às entradas e examinando os respectivos sinais de saída. A figura 43 demonstra um diagrama de blocos indicando a representação do processo. O *testbench* encapsula o controlador de estímulos, os resultados corretos e o DUT, além de conter os sinais internos que fazem as conexões adequadas entre os módulos. O controlador de estímulos envia os sinais de entrada ao DUT. O módulo DUT responde aos sinais de entrada e produz os resultados de saída. Finalmente, uma função interna ao *testbench* que compara os resultados do DUT com os resultados corretos esperados, pode reportar eventuais discrepâncias.

Existem muitas maneiras de se projetar um *testbench*, mas os modos mais comuns são:

- *Apenas estímulos*: Contêm apenas o controlador de estímulos e o DUT, mas não contém nenhum resultado de verificação;
- *Banco de testes completo*: Contém o controlador de estímulos, os resultados corretos e o comparador dos resultados.



**figura 43** Diagrama de um *TestBench* [62].

O método de simulação utilizado nesta dissertação foi o de *Apenas estímulo* pois apesar de termos calculado e conhecermos os valores para os resultados esperados dos sinais, o processo de verificação e comparação não estava integrado ao *testbench* e foi feito visualmente, sinal por sinal e etapa por etapa.

## **CAPÍTULO 4. METODOLOGIA, MATERIAIS E MÉTODOS**

Neste capítulo serão discutidas todas as soluções projetadas para resolver o problema de busca e localização de imagens invariante à escala e rotação (filtros Cifi e Rafi) em *hardware*. Serão discutidas todas as estruturas dos módulos matemáticos, módulos de leituras dos dados da memória e estruturas de registradores para o projeto em *pipeline* buscando garantir o máximo desempenho do sistema.

Os únicos códigos em VHDL de terceiros usados neste projeto (portanto não desenvolvidos neste trabalho) são aqueles para o cálculo da raiz quadrada e da divisão. Todos os demais arquivos VHDL podem ser implementados em qualquer FPGA (dependendo naturalmente da capacidade de cada modelo de dispositivo). Além disso, cada fabricante de FPGAs possui módulos gratuitos e prontos para o uso, para os cálculos da raiz quadrada e da divisão.

### **4.1. OBJETIVOS DA IMPLEMENTAÇÃO EM HARDWARE**

Para atingir ambos requisitos, de flexibilidade e desempenho, decidimos utilizar uma linguagem em alto nível, a linguagem C, para criar automaticamente todos os arquivos em VHDL necessários em todas as etapas de desenvolvimento propostas. A flexibilidade é garantida pela parametrização do código em C que então, gera códigos em VHDL específicos para os parâmetros inseridos mas totalmente otimizados, com uma quantidade eficiente de *pipelines*, permitindo elevadas frequências de operação.

Como mencionado, o objetivo mais importante desta implementação é o de maximizar a performance do algoritmo de busca, permitindo o maior desempenho na classificação de cada pixel da imagem de alta resolução como “pertencente” ou “não-pertencente” à imagem de máscara.

## 4.2. SOLUÇÃO EM *HARDWARE*

Nos tópicos seguintes são apresentados os projetos da solução final para os módulos em *hardware* para os filtro Cifi e Rafi. Ao optarmos pela estratégia de geração de códigos VHDL para garantir flexibilidade, pudemos ao mesmo tempo evitar que tal flexibilidade fosse também deixada a cargo do *hardware*. Isto implicaria fatalmente em um aumento desnecessário da lógica de todo o projeto e na diminuição drástica da frequência de operação de todo o sistema, pois além de todo o processamento realizado com os módulos projetados neste trabalho, seria necessário utilizar módulos em *hardware* para os cálculos de coordenadas dos círculos e raios, por exemplo.

Existem códigos em *hardware* disponíveis para este tipo de cálculo, mas ao utilizá-los não seria possível ao sistema atual operar em *pipeline*. Todos estes aspectos inviabilizariam assim, o desempenho do sistema de classificação (pertencente ou não-pertencente) alcançado.

Para os pontos mais críticos deste desenvolvimento, foram realizadas muitas pesquisas e análises para o projeto do *hardware* com a maior eficiência possível. Os pontos mais críticos para o desenvolvimento foram:

- A leitura de dados para a janela de processamento das médias;
- As somatórias dos pixels;
- A divisão para o cálculo das médias;
- As correlações e seus valores intermediários.

Todos estes tópicos são importantes no processamento matemático dos dois filtros.

### 4.2.1. SOLUÇÃO PARA O FILTRO CIFI E O FILTRO RAFI

A figura 44 demonstra os diferentes módulos da arquitetura do filtro Cifi em *hardware*. São também apresentadas as interconexões entre os filtros Cifi e Rafi.

O sistema completo de processamento, sem indicar os módulos de testes, ensaios e validação, é composto pelos módulos listados abaixo. Ainda que todos estes módulos tenham sido projetados e diagramados de modo a garantir o máximo desempenho, somente os módulos em negrito foram totalmente implementados, simulados e validados.

- Módulo de Leitura de Dados da Memória.
- **Controlador de Leitura/Escrita dos dados no filtro Cifi.**
- **Filtros Cifi e Rafi.**
  - **Módulo de Cálculo das Médias.**
    - **Janela Configurável de Processamento (CWP).**
      - Grande Árvore de Somadores.
    - **Divisão para a média.**
  - **Módulos de Correlações.**
    - **Árvores de Somadores Específica.**
    - **Módulos de Raiz Quadrada.**
    - **Módulo de Divisão.**
  - Módulo de Interconexão Entre os Filtros.
    - FIFO de Interconexão (Instanciação para cada pixel transferido).
  - **Módulo de Comparação das Correlações.**

É importante ressaltar que o projeto de cada módulo deve ser tal que permita seu processamento sem impactar no objetivo final de classificação de cada pixel da imagem a analisar  $A$ , a cada ciclo de relógio.

O Módulo Controlador de Leitura/Escrita abastece o sistema com os dados lidos da memória na ordem adequada e em paralelo de modo que a cada ciclo de relógio uma coluna completa é escrita na janela de processamento. Para simplificar o sistema, os dados só podem ser lidos da direita para a esquerda. Portanto, o controlador deve garantir, ao varrer a imagem  $A$ , que a janela de processamento foi completada com dados antes de disparar o processamento através do sinal de comando “MakeSum”.

Para uma janela de processamento com resolução de  $n \times n$  pixels, a cada linha,  $n$  colunas deverão ser lidas antes que o comando de iniciar a somatória (“MakeSums”) seja reiniciado.

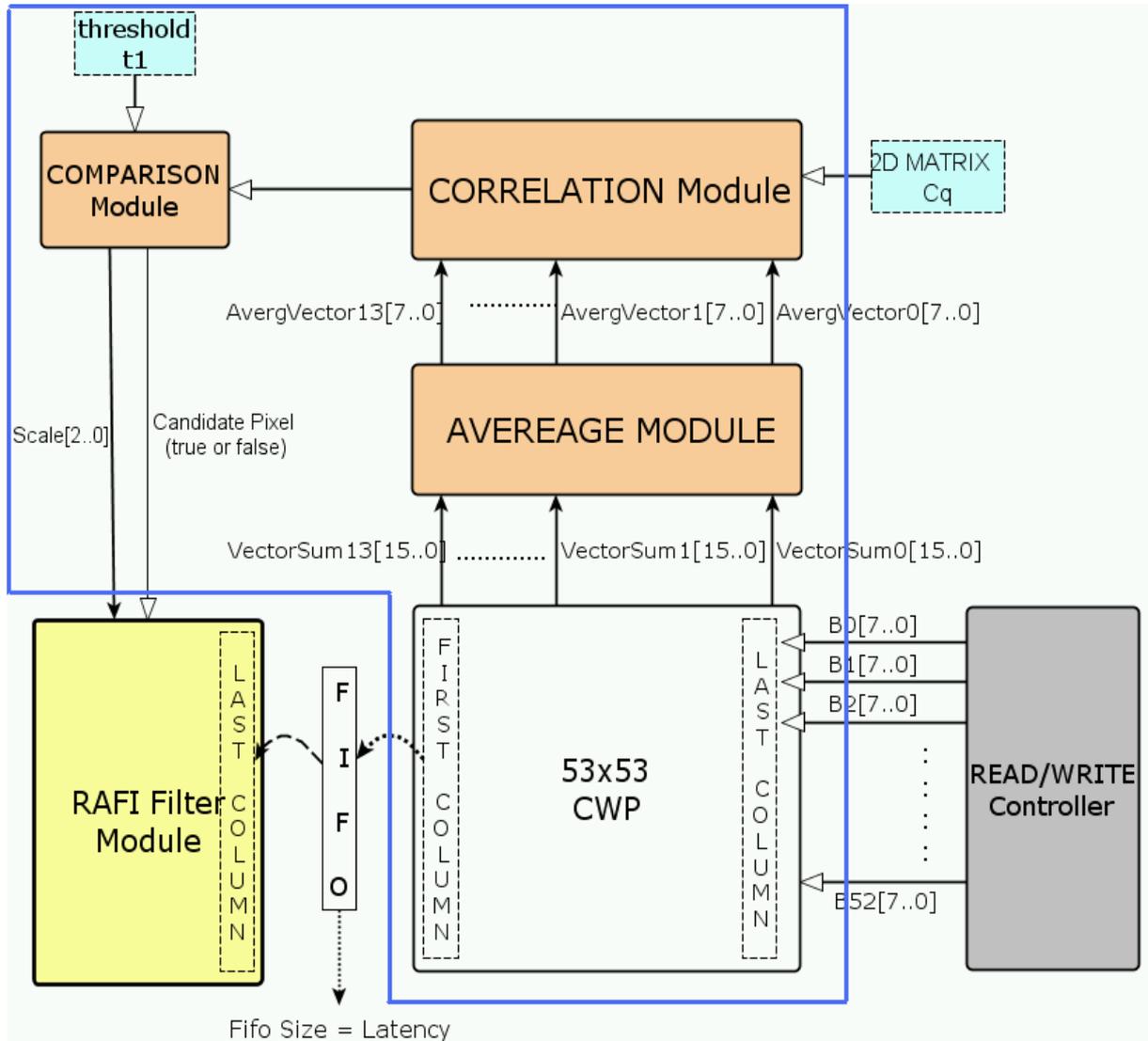
A figura 44 apresenta um diagrama de blocos dos principais módulos, salientando a interconexão entre os filtros Cifi e Rafi. Os módulos que compõem o filtro Cifi estão indicados na área delimitada em azul.

A FIFO recebe os dados dos pixels do Filtro Cifi. Os dados da FIFO são de 8 bits mas ela possui a resolução na horizontal (ou número de posições de 8 bits) correspondente à latência dos módulos seguintes à somatória: Módulo de Média, Módulo de Correlação e Módulo de Seleção/Comparação com Limiar.

Esta arquitetura garante que o Módulo Rafi, o segundo filtro, receba a coluna de dados e comece seu processamento assim que obtiver do filtro Cifi, a primeira indicação se o pixel é ou não um candidato do primeiro grau.

O Módulo de Comparação das Correlações recebe os coeficientes de correlação e seleciona, em *pipeline* (para garantir uma alta frequência de operação) aquele coeficiente de maior valor. O maior valor de correlação é então comparado com o valor de limiar. Se o maior coeficiente de correlação calculado for maior ou igual ao limiar então os sinais de saída deste Módulo de Comparação indicarão ao filtro Rafi que o pixel processado é um pixel candidato e seu provável fator de escala.

Como indicado, na abordagem proposta, a primeira coluna do Módulo Janela de Processamento (no filtro Cifi) é transferida para o próximo filtro (filtro Rafi) imediatamente após o cálculo da primeira correlação do filtro Cifi. Assim, não é necessário ao filtro Rafi aguardar o término do processamento Cifi para começar o processamento dos dados. Ambos filtros processam simultaneamente um pixel a cada ciclo de relógio, com a latência correspondente, de modo que a cada ciclo de relógio, um pixel é rotulado como pixel “candidato” ou “não-candidato”.



**figura 44** Interconexões entre os filtros Cifi e Rafi. Em detalhe na marcação indicada, o filtro Cifi.

Seguindo o fluxo de dados ainda na figura 44, após o cálculo das somas dos pixels em tons de cinza na Janela de Processamento, o módulo de cálculo das médias divide cada soma pelo número de pixels de cada círculo. Na seqüência, usando os dados pré-calculados da matriz  $C_q$ , o maior valor de correlação é selecionado. Finalmente, nós comparamos o resultado da maior correlação calculada com o dado valor de limiar com a finalidade de inferir se o pixel processado possui alguma chance de pertencer à imagem de máscara buscada. O provável valor de escala é aquele cujo valor de correlação foi o maior das correlações encontradas para cada escala processada.

### **4.2.1.1. HARDWARE PARA O CÁLCULO DAS MÉDIAS**

O cálculo das médias é a primeira etapa do processamento para os filtros Cifi e Rafi mas envolve diversos aspectos de projeto que devem ser resolvidos para viabilizar este cálculo de modo eficiente. Projetamos uma janela de processamento configurável ou “*Configurable Window Processor*” (CWP) que deverá “percorrer” a imagem *A*, calculando as médias dos círculos ou dos raios. O aspecto mais fundamental é o de como os dados serão escritos na janela de processamento e como a matriz CWP irá varrer a imagem *A*.

O ponto seguinte é de como serão feitas as somas da grande quantidade de pixels. No caso do filtro Rafi, os limites das quantidades de pixels contidos nas linhas radiais a serem somados dependerão do fator de escala calculado pelo filtro Cifi e informado ao filtro Rafi.

A etapa final é o cálculo para a média. Como a divisão é um cálculo complexo em *hardware* e que envolve certa latência, e ainda, como diversos modelos de FPGAs possuem multiplicadores DSP integrados ao dispositivo, decidimos realizar esta divisão final através de uma multiplicação. As somas dos valores dos pixels que seriam divididas pela quantidade de pixels nos círculos (filtro Cifi) ou raios (filtro Rafi), são multiplicadas pelo inverso da quantidade de pixels do respectivo círculo ou raio. Este valor a ser multiplicado está em ponto fixo. A análise dos valores para esta solução comprovou que o erro desta simplificação é desprezível. Isto é discutido oportunamente mais adiante.

#### **4.2.1.1.1. LEITURA DOS DADOS**

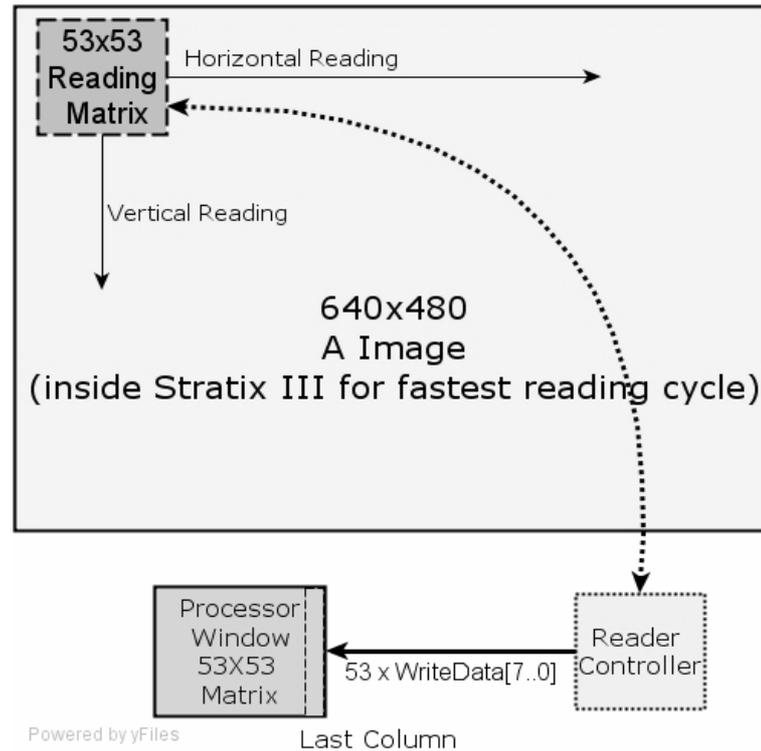
Nosso sistema utiliza um módulo em *hardware*, chamado de CWP que consiste em uma matriz de processamento implementada com o propósito de realizar as somatórias dos pixels dos círculos ou raios para todos os pixels da imagem *A*. Um trabalho anterior [35] utiliza a mesma expressão, com um propósito diferente: a janela de processamento CWP proposta possui resolução de 7x7 pixels

e é configurável no sentido de que pode assumir diferentes funcionalidades. Em nosso caso, a janela CWP pode ter sua resolução configurada de 23x23 pixels até 53x53 pixels. Mesmo estes limites podem ser facilmente modificados.

Para que os dados sejam lidos da imagem *A* e corretamente transferidos para a matriz CWP, foi projetado um módulo controlador. Inicialmente, projetamos e implementamos um sistema que permitisse à matriz CWP se deslocar tanto para a direita quanto para baixo, ao percorrer a imagem *A*. Durante os testes de síntese e roteamento, esta configuração produziu um *hardware* pouco compacto, com um número exagerado de elementos lógicos (mais de 77.000 elementos lógicos apenas para o módulo de somatórias) e baixa frequência de operação (em torno de 55 MHz).

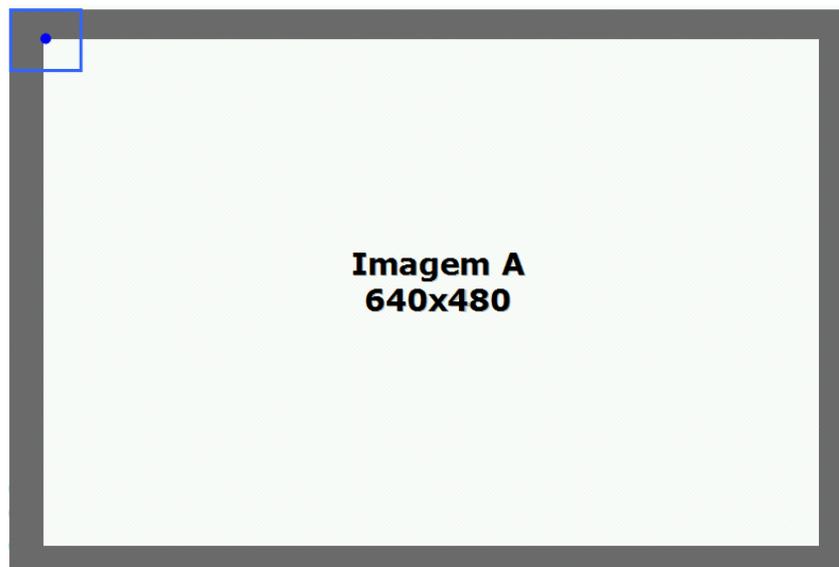
A solução foi simplificar o sistema de escrita dos dados na matriz CWP que passaram a ser feitos somente da direita para a esquerda. A quantidade de elementos lógicos necessários para implementar o *hardware* da nova configuração diminuiu cerca de 8 vezes. Verificou-se que a frequência máxima de operação do sistema também aumentou com a compactação do módulo. O revés foi que para cada mudança de linha, as primeiras 53 (para uma matriz CWP com resolução de 53x53 pixels) novas colunas deverão ser lidas antes de reiniciar o processo de somatória. Cada leitura e escrita de uma coluna com 53 pixels é feita em um ciclo de relógio. Atingida esta latência de 53 leituras e escritas (53 ciclos de relógio), todos os ciclos seguintes, para a mesma linha processada, são feitos em um ciclo de relógio.

A figura 45 representa a arquitetura do *hardware* responsável por ler os pixels da imagem *A* para a janela de processamento CWP. Uma coluna completa é escrita na janela CWP a cada ciclo de relógio. Os pixels são então escritos na janela CWP de apenas uma maneira, da direita para a esquerda.



**figura 45** Arquitetura para leitura de dados.

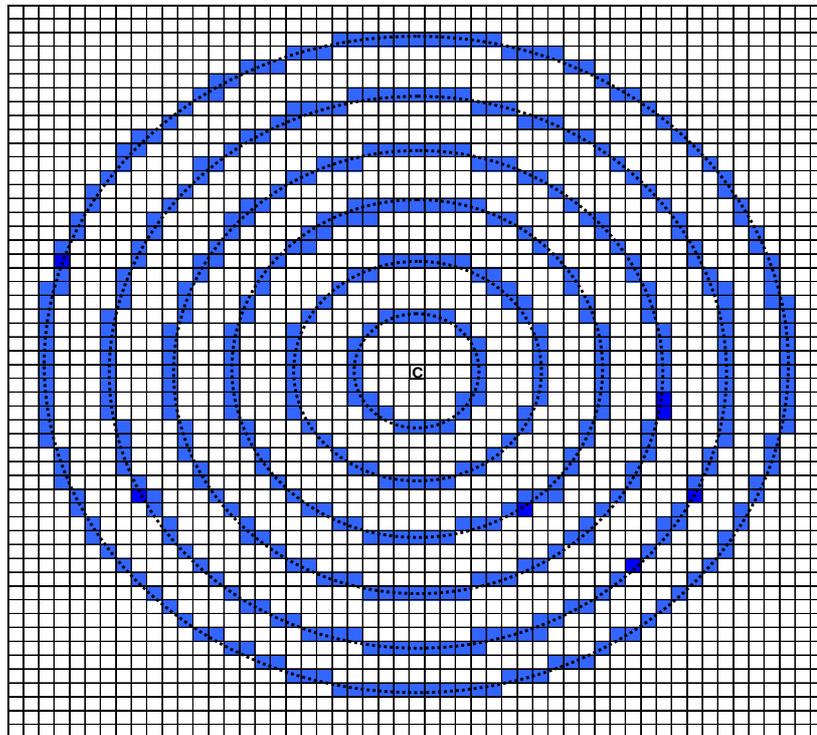
Exceto por uma fina borda na imagem *A*, todos os pixels são processados, como indicado na figura 46. A espessura desta borda não-processável se refere à metade da resolução da matriz CWP pois o primeiro pixel processável é justamente o pixel central da matriz CWP que foi lido da imagem *A*. Na mesma figura, nota-se também a indicação do primeiro pixel a ser processado, dentro da posição inicial da janela de processamento.



**figura 46** Região em branco interna indica pixels processáveis para uma imagem *A* de 640x480.

### 4.2.1.1.2. JANELA DE PROCESSAMENTO E ÁRVORE DE SOMADORES

A representação da janela de processamento para o filtro Cifi está indicada na figura 47 para uma janela de processamento configurada para a resolução de 53 pixels (máximo especificado para este projeto) e contendo uma quantidade de 6 círculos mais o pixel central (o máximo especificado para este projeto é de 14 círculos para uma matriz CWP de 53x53). Cada um dos quadrados preenchidos na figura indica o pixel que deve ser somado e que juntos, integrarão os processos de somatórias.



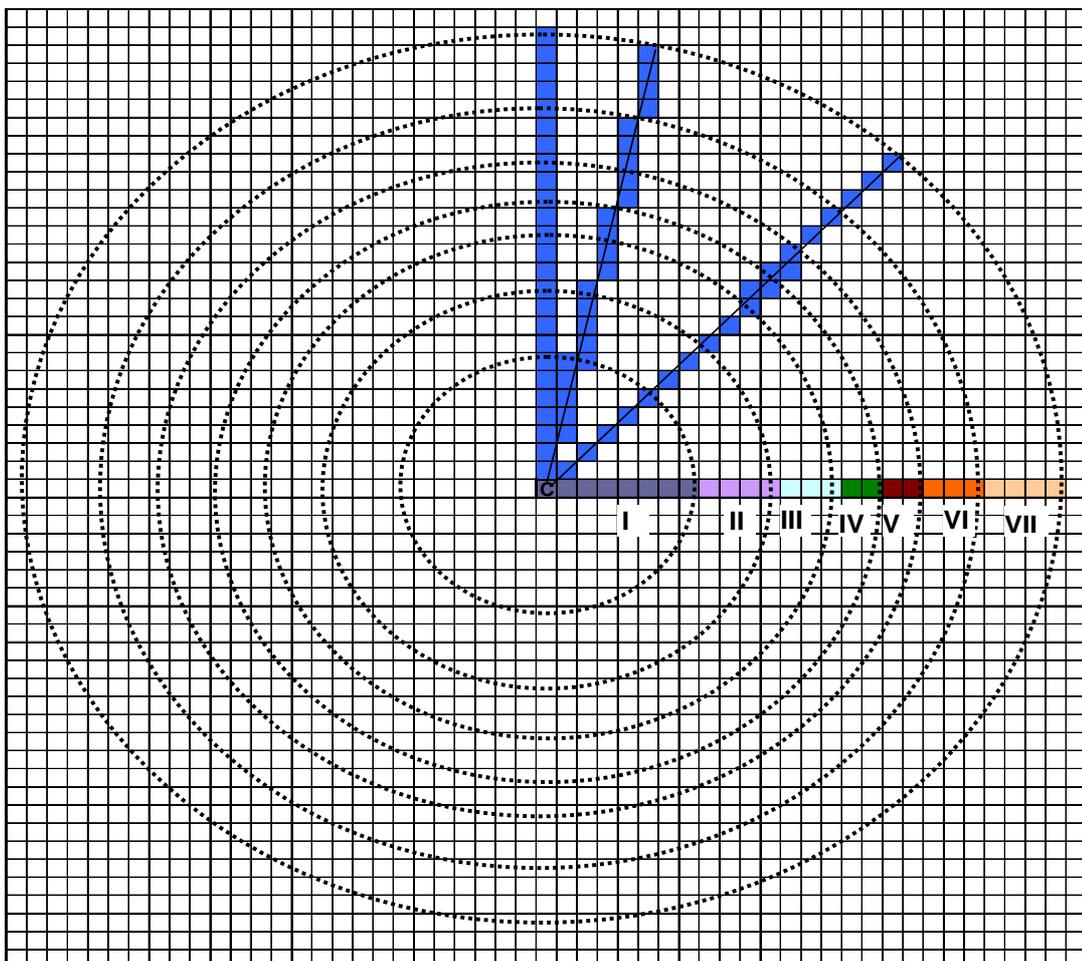
**figura 47** A janela de processamento para o filtro Cifi.

Para o filtro Cifi, todos os pixels da imagem *A* (exceto pela borda) são processados. Já para o filtro Rafi, somente serão processados os pixels candidatos do primeiro grau detectados pelo filtro Cifi.

A figura 48 representa a janela de processamento CWP para o filtro Rafi que difere em sua estrutura do filtro Cifi, no sentido de que a quantidade de pixels (tamanho do raio) a ser somado dependerá do valor da escala do pixel candidato detectado pelo filtro anterior, o filtro Cifi.

Os sete círculos da figura 48 correspondem às resoluções das sete escalas de máscara utilizadas no algoritmo Ciratefi para este trabalho. Os sete círculos estão “plotados” na imagem sobre a matriz de processamento CWP para o filtro Rafi que possui resolução de 53x53 pixels. Estes parâmetros (resolução e número de escalas) estão especificados para o caso mais complexo. Um aspecto importante que pode ser verificado através da mesma figura é que para diferentes ângulos, a quantidade de pixels contida nos raios pode variar. O programa gerador de códigos VHDL deve gerenciar estes pontos de modo a manter o comprimento em pixels igual para todos os raios.

As regiões de I a VII da figura correspondem aos sete diferentes trechos de cada linha radial a ser somado. Conforme a escala de cada pixel candidato processado, as respectivas regiões de somadores (de I a VII) serão tomadas pelo processo de somatórias (Árvore de Somadores). O módulo responsável por esta seleção é o módulo “Decoder” conforme ilustrado mais adiante na figura 50.



**figura 48** A janela de processamento para o filtro Rafi.

Os parâmetros que impactam na complexidade da geração do código para o cálculo de Rafi são o número de escalas e a resolução das escalas envolvidas. As resoluções das escalas definem os comprimentos das linhas radiais a serem considerados para o cálculo das médias das linhas radiais.

A fim de se maximizar a performance do *hardware* projetado, nós fizemos uso intenso de sistemas de processamento em *pipeline*, registrando todos os resultados intermediários de modo a obter, com esta estratégia, um resultado final do algoritmo Ciratefi (a indicação se o pixel é ou não um pixel candidato) a cada ciclo de relógio.

O processo de somatórias é feito sempre somando-se dois pixels dois-a-dois, em paralelo e registrando os resultados antes da soma seguinte. A representação desta configuração é uma árvore de somadores. Para o filtro Cifi, a árvore de somadores está representada na figura 49. É importante notar que todas as somas passam por registradores (indicados pelos retângulos com “R”). Estes registradores são os diferentes níveis de *pipeline* (indicados por  $L_n$  para  $n$  variando de 1 a 5 para o caso indicado).

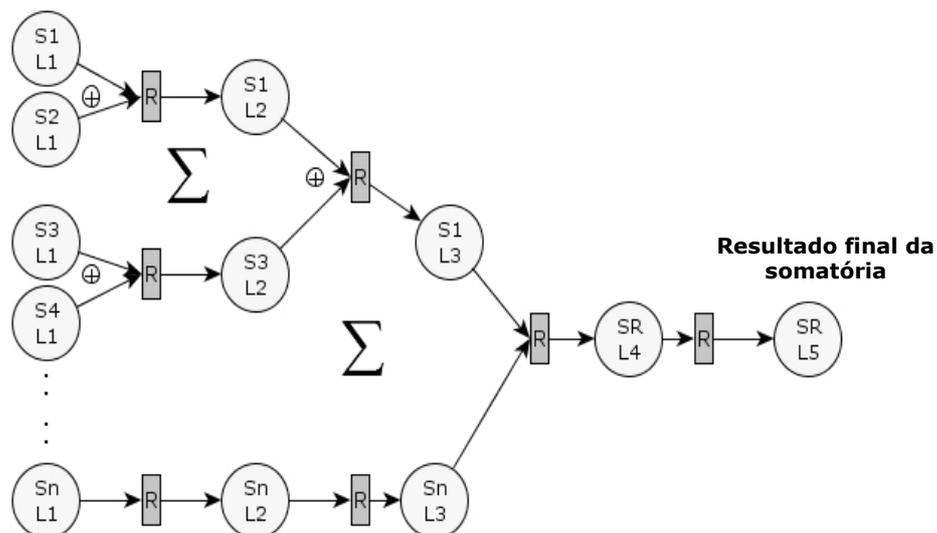
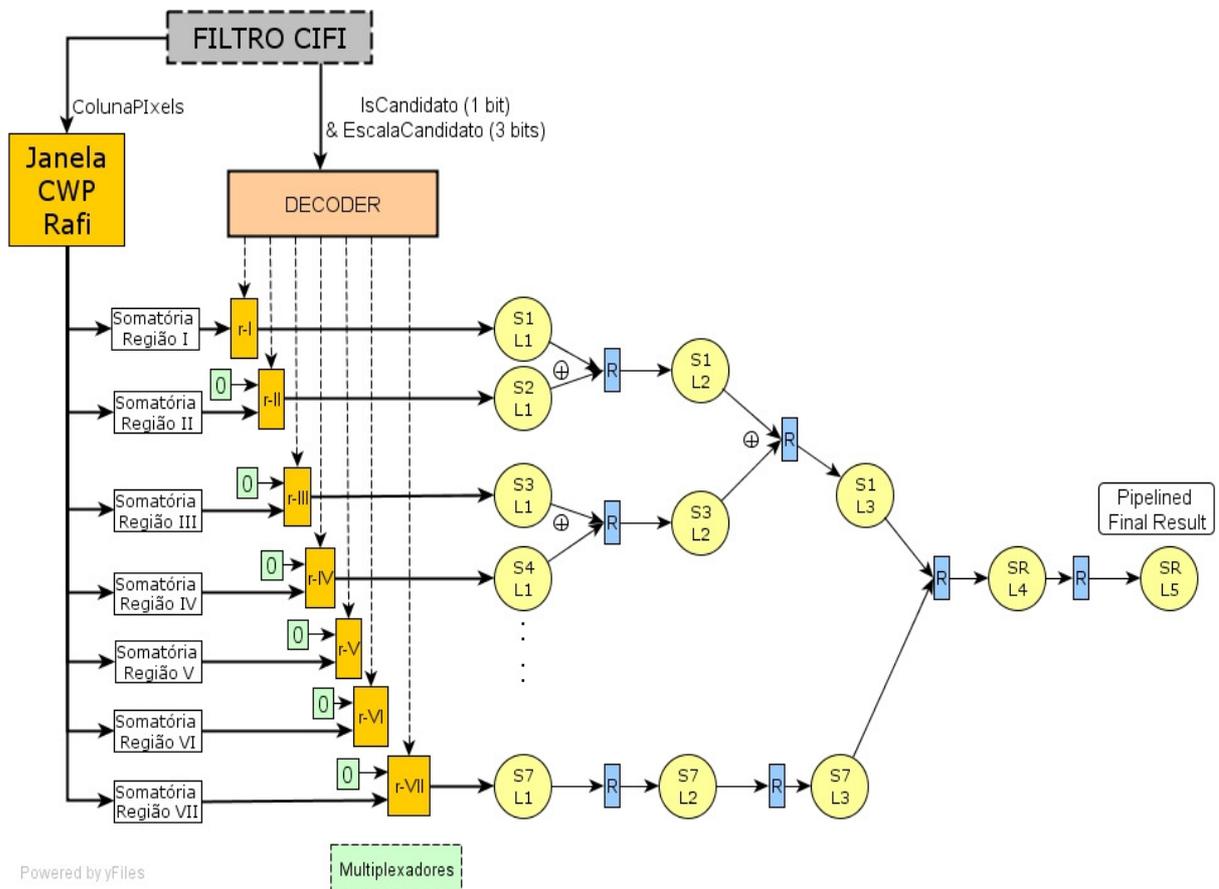


figura 49 Árvore de somadores em *Pipeline* para o filtro Cifi.

É importante ainda salientar que são instanciadas tantas árvores de somadores quanto o número de círculos na matriz CWP do filtro Cifi, de modo que as somatórias de todos os círculos ocorrem de forma paralela. Como as somatórias dos círculos centrais terminam antes da somatória do círculo da borda (diferentes quantidades de pixels por círculo, ou seja, diferentes latências), são criados registradores de “atraso” para os resultados que terminem antes, a fim de que ao final, tenha-se todos os resultados prontos e sincronizados. Esta abordagem de

sincronização é fundamental para garantir o processamento correto, em paralelo, de todos os dados. Além disso, o impacto na frequência máxima de operação do sistema tende a ser até ampliado pelo uso de registradores, embora aumente-se a quantidade de elementos lógicos necessários na síntese do *hardware*.

O processo de somatória para o filtro Rafi é semelhante ao do filtro Cifi. No entanto, como mencionado, para o caso do filtro Rafi, deve-se levar em conta que os limites para a soma dos pixels dos raios pode ser diferente segundo a escala mais adequada para representar determinado pixel candidato. Para levar este aspecto em consideração basta apenas implementar multiplexadores dispostos antes da árvore de somadores, segundo a figura 50.



**figura 50** Árvore de somadores em *Pipeline* para o filtro Rafi com o módulo “Decoder” .

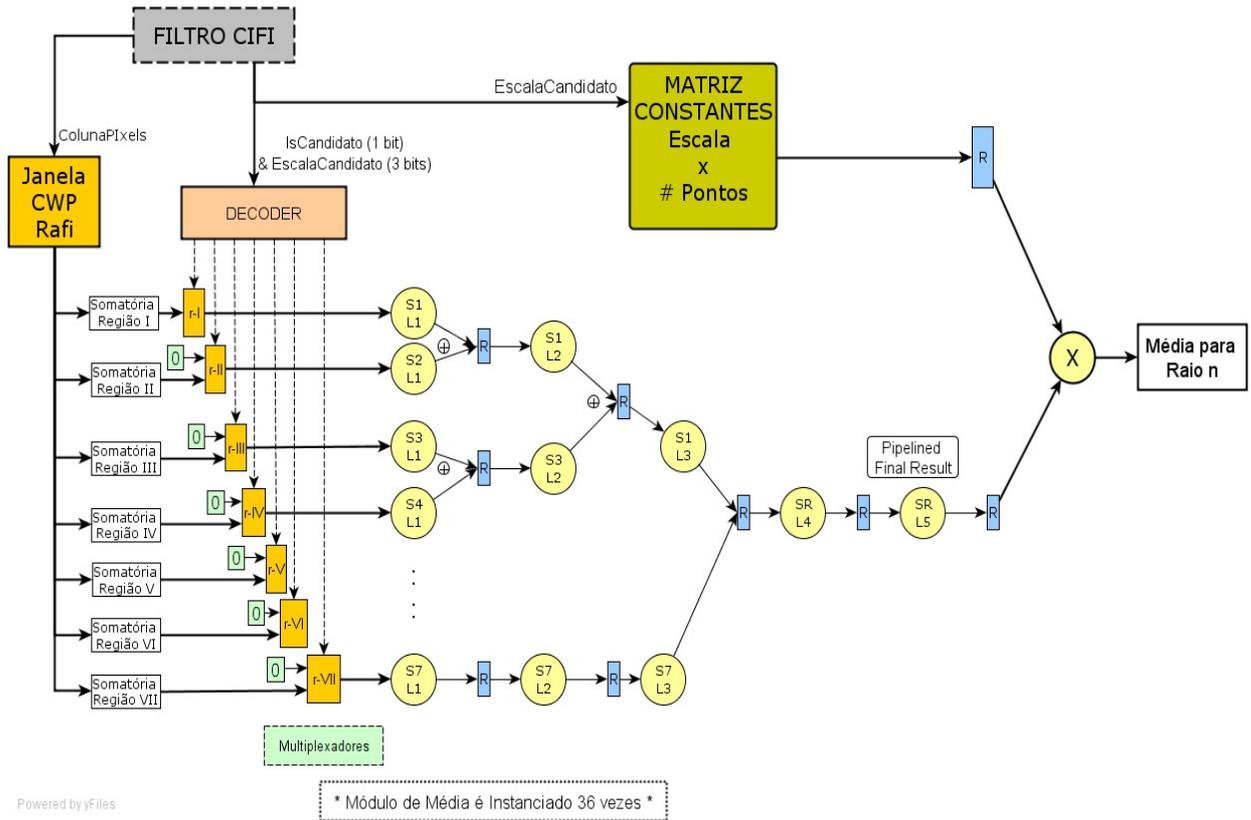
A função destes multiplexadores é a de selecionar quais dados entrarão no processo de somatórias. Caso a escala de determinado pixel candidato do primeiro grau não seja a máxima, então serão selecionados valores “zero” em tantos multiplexadores quanto necessários para que se corresponda a região dos raios de interesse com o valor de escala selecionado. O módulo responsável por fazer esta seleção é o módulo “Decoder” incorporado ao processo.

Ainda na figura 50, deve-se assumir que para cada bloco de somatória por região ilustrado (de 1 até 7, correspondendo ao número de escalas), é implementado uma árvore de somadores que executa em paralelo a soma dos pixels referentes somente àquela região. Ao se utilizar 36 raios, deverão se instanciadas 36 árvores de somadores que irão operar em paralelo na janela de processamento CWP do filtro Rafi.

### **4.2.1.1.3. CÁLCULO DA DIVISÃO PARA A MÉDIA**

A última etapa necessária para o processo de cálculo das correlações, é o cálculo das médias. Como mencionado, pelo fato de que os multiplicadores já estão incorporados aos próprios dispositivos FPGAs em silício, implementamos os cálculos da divisão através da multiplicação pelo inverso, em ponto-fixo, da quantidade de pixels presentes nos diferentes círculos ou raios.

Mais uma vez, no caso do filtro Rafi, pelo fato de que a quantidade de pixels presentes em cada raio pode variar entre as 7 diferentes escalas, foi necessário acrescentar à estrutura de cálculo das médias do filtro Rafi, um módulo do tipo *lookup table*, conforme apontado na figura 51 que contém o valor da quantidade de pixels de cada um dos 36 raios diferentes, sendo que para cada raio, haverá o valor da quantidade de pixels para as 7 regiões possíveis (por causa das 7 escalas processadas). Assim, conforme a escala do pixel candidato, a respectiva constante será utilizada. Esta constante refere-se então, ao inverso do número de pixels para o raio de comprimento correspondente a escala do pixel candidato.



**figura 51** Árvore de somadores em *Pipeline* para o filtro Rafi com o módulo “Decoder” e o processo de cálculo da média.

### 4.2.1.2. **HARDWARE PARA O CÁLCULO DAS CORRELAÇÕES**

Para o cálculo dos coeficientes de correlação entre dois vetores  $x$  e  $y$ , nós utilizamos a equação 3:

$$Corr(x, y) = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}} \quad (3)$$

Onde:

$n$  : Número de círculos presentes nas diferentes escalas para o filtro Cifi ou o número de raios para o Filtro Rafi (igual a 36 para qualquer escala);

$X_i$  : Média das somas dos pixels processados pelo *hardware* ao percorrer a imagem  $A$ ;

$Y_i$  : Média das somas dos pixels da imagem de máscara (obtidas via pré-processamento). Valores fixos.

Os tópicos seguintes cobrem toda a metodologia para implementar todos os cálculos matemáticos envolvidos na equação da correlação citada.

#### 4.2.1.2.1. **CÁLCULOS INTERMEDIÁRIOS**

A primeira etapa do projeto do *hardware* para o cálculo da correlação foi analisar todos os cálculos matemáticos envolvidos. Os cálculos mais complexos que são a raiz quadrada e a divisão serão discutidos em tópicos específicos.

Em seqüência, foi analisada a dependência dos cálculos na correlação de modo que todas as operações envolvidas fossem distribuídas em 6 etapas separadas por ordem de dependência. O resultado desta análise é demonstrado na figura 52.

Os cálculos que não envolvem raiz quadrada e divisão possuem latência menor e, portanto, atenção maior deve ser dada à sincronização destes resultados intermediários. Ainda na mesma figura, nota-se que o numerador (da divisão que é a última operação para o cálculo da correlação) é calculado na etapa 3, muito antes do denominador. A seta pontilhada na mesma figura indica este termo do numerador que deve manter-se válido até a divisão final.

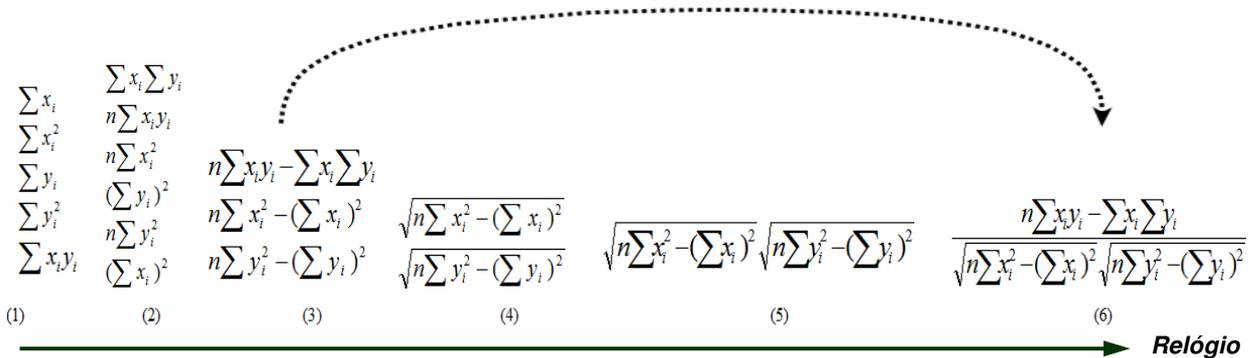


figura 52 Cálculos intermediários para a computação da correlação.

A figura 53 apresenta os cálculos intermediários da correlação para uma análise dos tamanhos dos sinais envolvidos (em bits), para o pior caso do filtro Cifi. Esta etapa de análise é importante para a definição da largura em bits de todos os sinais.

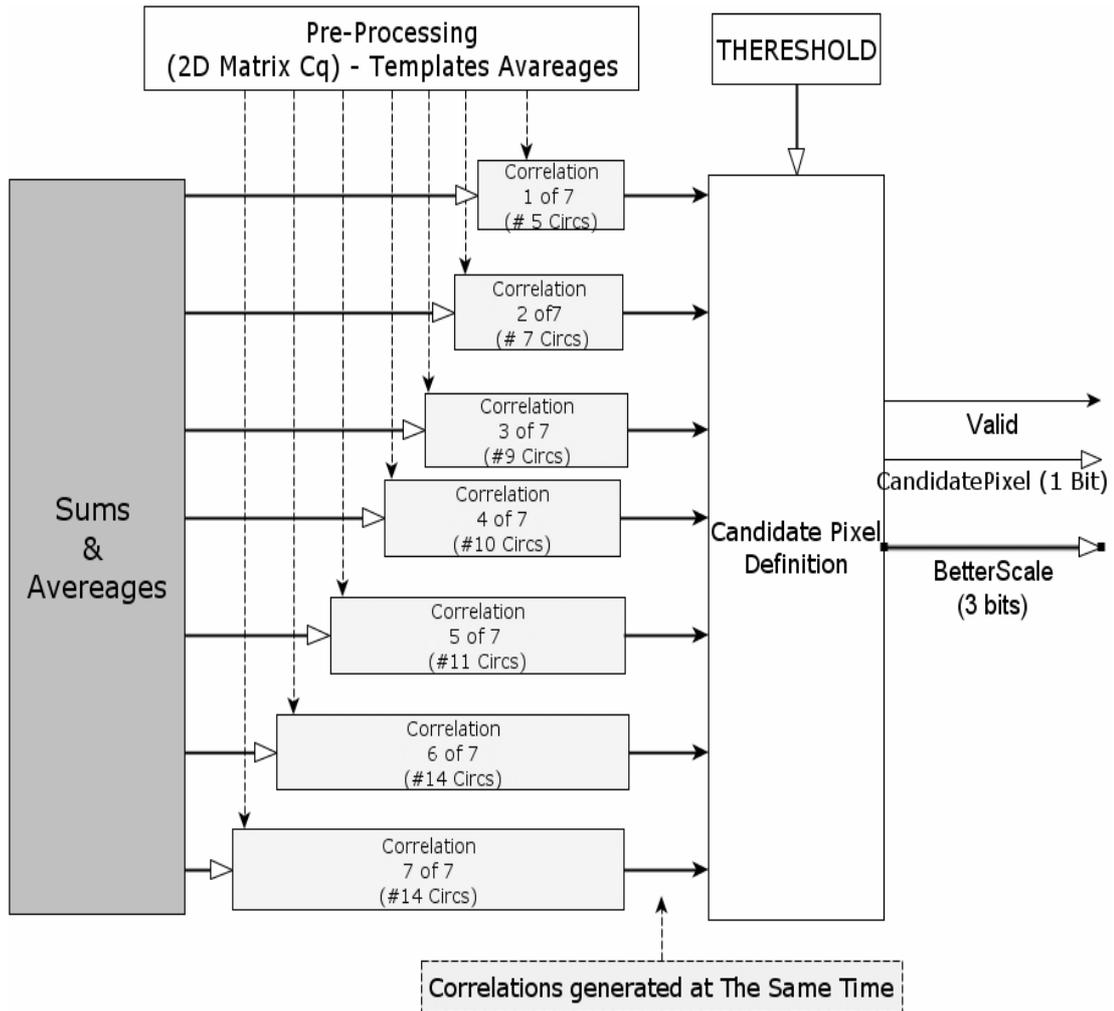
Valor	Número de bits	Somadores	Bits Somadores	Multiplicadores	Bits Mult.	Divisão	Bits Divisao
SUM(Xi*Yi)	20	13	16	13	8;8>16	0	0
SUM(Xi)	12	13	8	0	0	0	0
SUM(Yi)	12	13	8	0	0	0	0
SUM(Xi^2)	20	13	16	13	8;8>16	0	0
SUM(Yi^2)	20	13	16	13	8;8>16	0	0
n*SUM(Xi*Yi) [1]	24	0	0	1	20;04>24	0	0
(SUM(Xi)*SUM(Yi)) [2]	24	0	0	1	12;12>24	0	0
<b>([1] - [2]) [A]</b>	24	1	24	0	0	0	0
n*SUM(Xi^2)	24	0	0	1	20;04>24	0	0
n*SUM(Yi^2)	24	0	0	1	20;04>24	0	0
(SUM(Xi))^2	24	0	0	1	12;12>24	0	0
(SUM(Yi))^2	24	0	0	1	12;12>24	0	0
(n*SUM(Xi^2)-((SUM(Xi))^2)) [1]	24	1	24	0	0	0	0
(n*SUM(Yi^2)-((SUM(Yi))^2)) [2]	24	1	24	0	0	0	0
SQRT([1])	24	?	?	?	?	0	0
SQRT([2])	24	?	?	?	?	0	0
<b>(SQRT([1]) * SQRT([2])) [B]</b>	12	0	0	1	12;12>24	0	0
<b>([A] &lt;&lt; 16)/[B]</b>	0	0	0	0	0	1	40;24>16

figura 53 Análise dos sinais digitais envolvidos no cálculo completo da correlação

Os resultados das correlações são dados em ponto-fixa e, por isso há a indicação do deslocamento de 16 bits para o último termo, ainda na figura 53. Os resultados armazenados como sinais em ponto-fixa de 16 bits possuem os 16 bits para a parte fracionária, o que oferece resultados finais com resoluções (ou passos) de  $1/(2^{16})$  ou de 0,000015258790.

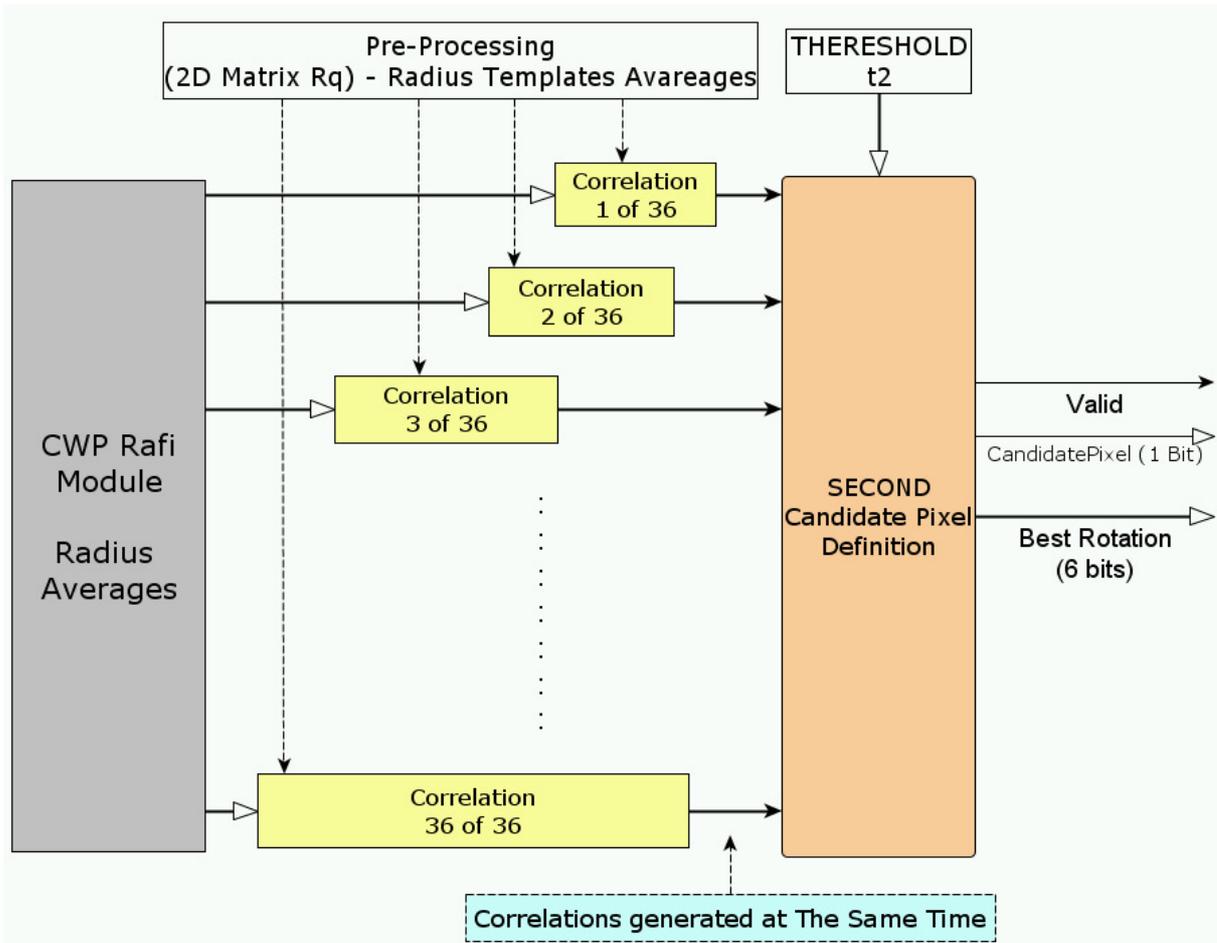
Apesar do Módulo de Correlação envolver cálculos matemáticos mais complexos em *hardware* e que foram projetados com diversos ciclos de latência, uma vez completados estes ciclos de latência, a cada ciclo de relógio será disponibilizado um resultado de correlação.

É por este motivo que nós implementamos tantos módulos de correlação quanto diferentes fatores de escala presentes no algoritmo (para o filtro Cifi). O diagrama de blocos da figura 54 demonstra este propósito, onde um módulo de correlação diferente é instanciado por cada máscara de escala. O artigo em [35] utiliza uma solução similar para fazer diversos cálculos matemáticos simultaneamente. O número de fatores de escala é configurável. Nosso programa em C gera módulos com até 7 diferentes fatores de escala.



**figura 54** Arquitetura para o cálculo da correlação em paralelo para o filtro Cifi.

De modo similar ao cálculo da correlação para o filtro Cifi, no caso do filtro Rafi, são instanciados 36 módulos de correlação em paralelo conforme a figura 55. A concepção é similar e o objetivo permanece o de obter um resultado de classificação a cada ciclo de relógio.



**figura 55** Arquitetura para o cálculo da correlação em paralelo para o filtro Rafi.

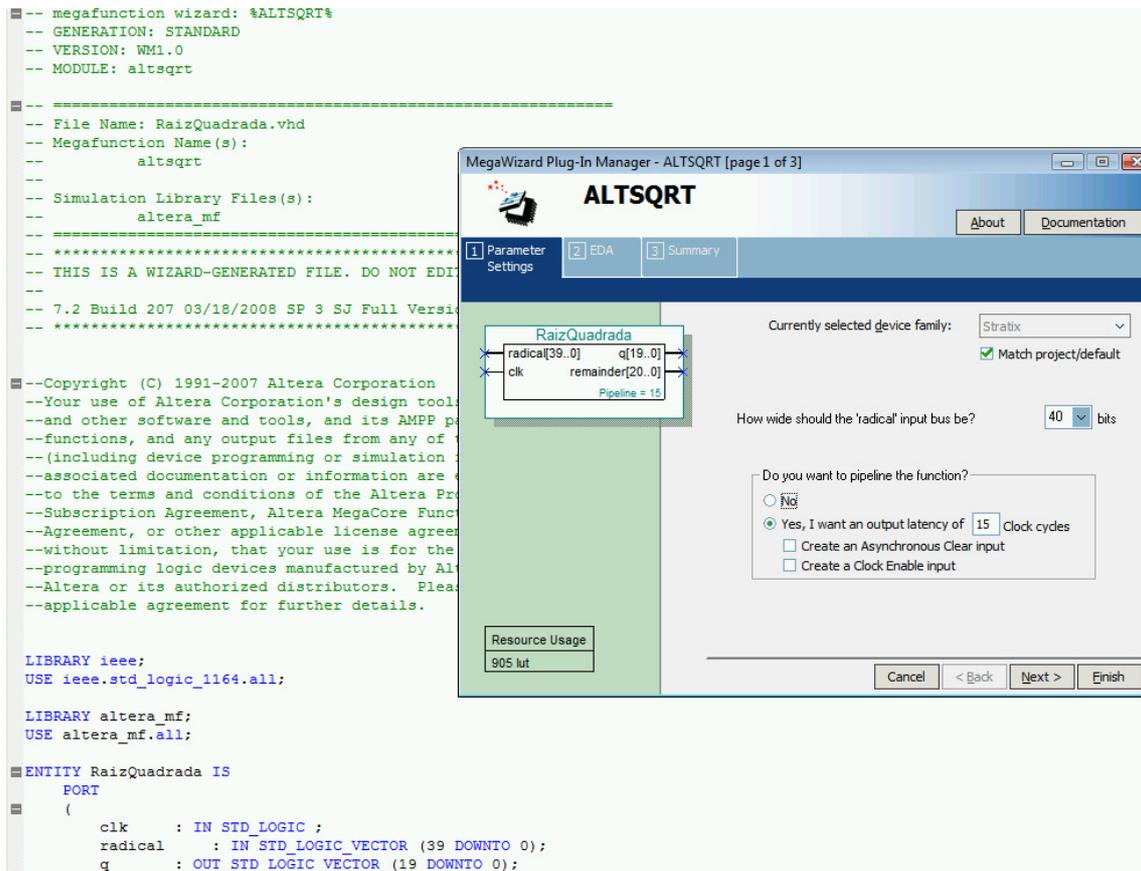
### 4.2.1.2.2. CÁLCULOS DE RAIZ QUADRADA

Para os cálculos envolvendo raiz quadrada, são utilizadas as bibliotecas já prontas e gratuitas, disponibilizadas pela Altera para este tipo de cálculo em *hardware* através da ferramenta *MegaWizard* (figura 56). Os demais fabricantes possuem ferramentas semelhantes para o mesmo cálculo.

Os códigos-fonte gerados por estas ferramentas não são disponibilizados. Porém, as ferramentas deste tipo geram um código VHDL de topo de hierarquia que instancia as bibliotecas apropriadas. Neste trabalho, de modo que todos os cálculos já estivessem incorporados dentro de um módulo aglutinado (correlação) onde se encontram todos os demais cálculos (multiplicação, registradores de sincronismo, e árvores de somadores), foi gerado um módulo em VHDL da raiz quadrada apenas para análise dos parâmetros para as bibliotecas. Assim, nosso programa em C pode gerar códigos compiláveis substituindo a ferramenta do fabricante (*MegaWizard* - Altera) que gera o código VHDL.

Como na etapa 4 da 0, em que devem ser calculadas duas raízes quadradas, são instanciados dois módulos de *hardware* de Raiz Quadrada dentro do mesmo Módulo de Correlação. Assim, cada Módulo de Correlação é composto por: dois módulos para o cálculo da raiz quadrada, um para divisão, um para o sistema de somatórias dos produtos e um para o gerenciamento dos sinais em *pipeline*. Esta abordagem permite que os cálculos ocorram em paralelo.

Ainda através da figura 56, nota-se que um dos parâmetros configuráveis é justamente a latência a ser usada para o módulo em questão. Após alguns testes com a síntese, roteamento e análise dos tempos de operação, percebeu-se que o valor ótimo para a latência (estágios de *pipeline* a serem criados) para os cálculos da raiz quadrada, eram iguais ao número de bits do radical. Valores abaixo deste irão diminuir o desempenho em termos de frequência de operação do módulo, e valores acima não tem nenhum efeito sobre a frequência ou podem até mesmo piorar o desempenho do módulo.



**figura 56** Em primeiro nível tem-se imagem da ferramenta *MegaWizard* com as opções de configuração para a raiz quadrada e ao fundo o código VHDL gerado.

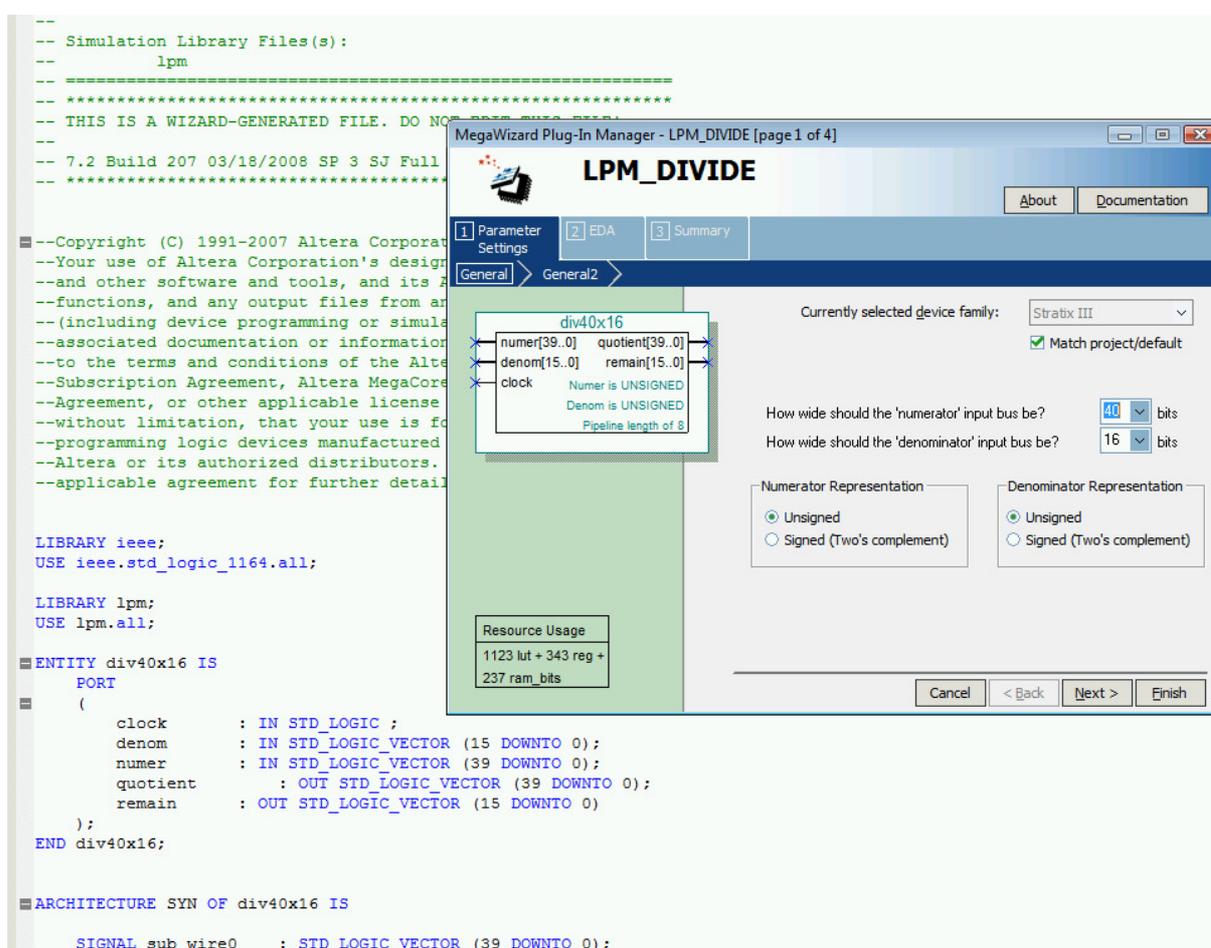
### 4.2.1.2.3. CÁLCULO DA DIVISÃO

A divisão é o último cálculo matemático feito para se obter o resultado final da correlação. A figura 57 apresenta ao fundo o código VHDL gerado pela ferramenta *MegaWizard* do software Quartus II da Altera, e em primeiro plano se encontra a tela para entrada dos parâmetros de configuração. Como mencionado, a análise do código VHDL criado pela biblioteca da Altera possibilitou ao programa gerador projetado e implementado neste trabalho, também a capacidade de configurar e gerar adequadamente o módulo de divisão com os parâmetros de configuração correspondentes à solução do algoritmo proposto.

O cálculo da divisão em *hardware* produz resultados em ponto-fixo e não em ponto flutuante. Os resultados da correlação em ponto flutuante são valores que variam de 0 até 1 para a correlação máxima. Como indicado, para o sistema

implementado neste trabalho o resultado final é dado em ponto-fixe do tipo “0.16” indicando que todos os 16 bits são atribuídos à parte fracionária, permitindo uma resolução de  $1/(2^{16})$ . Para isso, o numerador da fórmula de correlação  $n \sum x_i y_i - \sum x_i \sum y_i$  é deslocado de 16 bits para a esquerda (ou multiplicado por 65536) antes do cálculo da divisão. Assim, a partir do resultado do cálculo em *hardware*, basta dividir os valores obtidos da divisão, por 65536 para obter na forma fracionária o valor em decimal (entre 0 e 1) para a correlação calculada em *hardware*.

Portanto, em hardware o valor máximo para a correlação não será 1, mas será em ponto-fixe “FFFF” (hexadecimal) correspondendo a 0,9999847412109375 em valor decimal.



**figura 57** Em primeiro nível tem-se imagem da ferramenta *MegaWizard* com as opções de configuração para a divisão e ao fundo o código VHDL gerado.

#### 4.2.1.2.4. MÓDULO DE SELEÇÃO DA MAIOR CORRELAÇÃO

Neste tópico discutimos o projeto do módulo cujo objetivo é a seleção do valor de maior correlação e indicação final se o pixel é um candidato do primeiro grau com base no valor de limiar fornecido. Este valor de limiar deve ser convertido para ponto fixo do tipo “0.16” pois a comparação será feita também em *hardware*. Para isso basta tomar o valor em decimal, por exemplo 0,9 e multiplicá-lo por 65536. No caso, o resultado será de 58982,4. Mas com a truncagem, tomando somente o valor inteiro temos E666 (em hexadecimal com 16 bits).

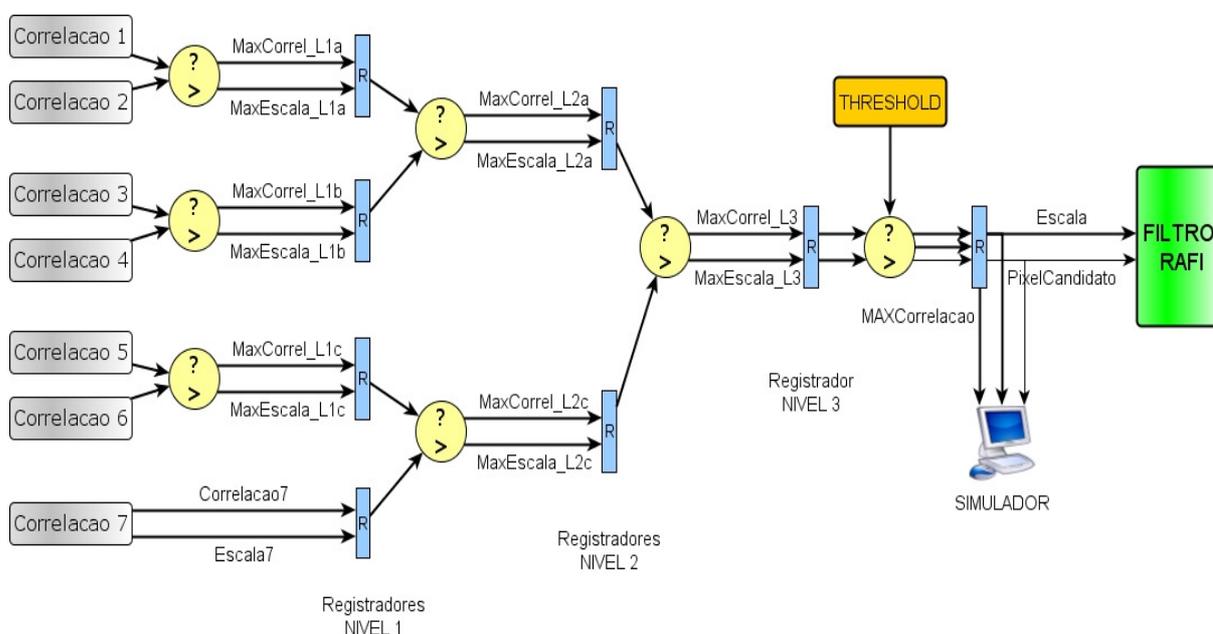


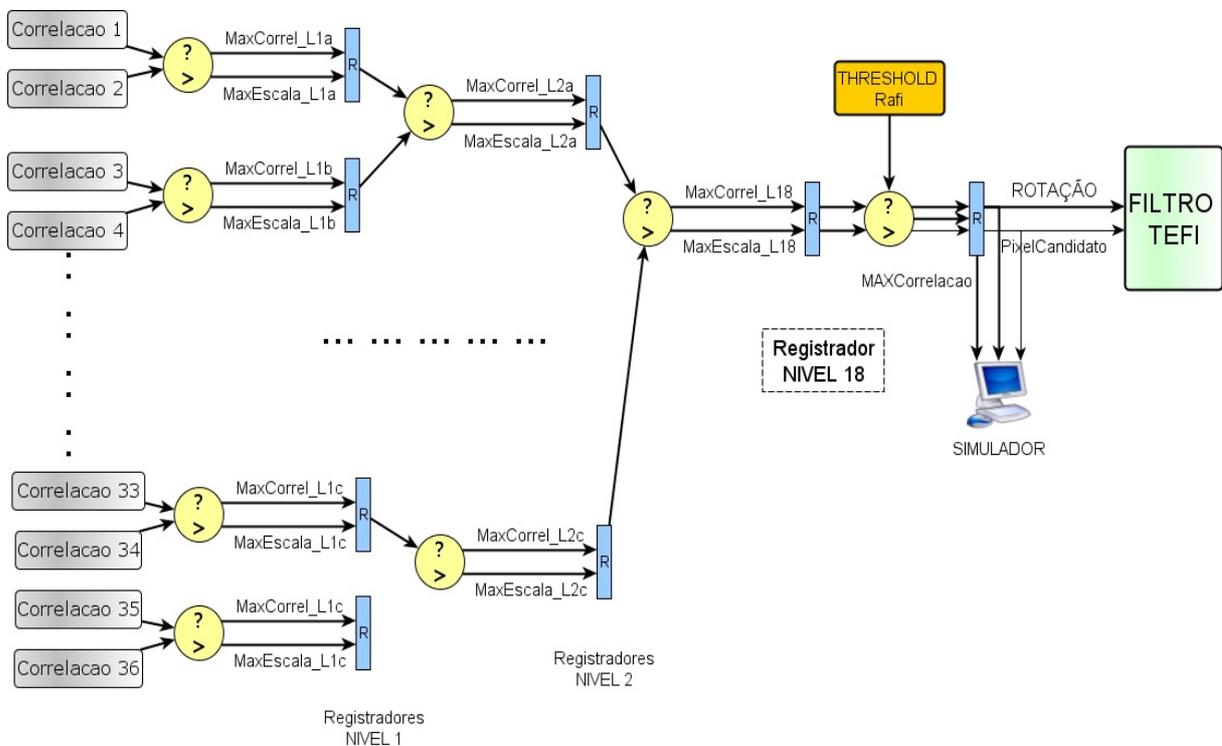
figura 58 Árvore de seleção de correlações para o filtro Cifi.

A figura 58 apresenta o diagrama para a árvore de comparadores em *pipeline* que, dado os valores de correlação de entrada (em 16 bits) seleciona: a maior correlação e o valor da escala da respectiva correlação.

É importante notar que o sinal indicando o valor da escala passa também por todo o fluxo de dados. Sinais que não possuem seu par (pois são 7 correlações para as 7 escalas diferentes para a configuração mais complexa) são registrados sem nenhum processamento, garantindo o sincronismo durante toda a comparação e permitindo que a cada ciclo de relógio, novos valores de correlação (para os pixels seguintes) entrem dentro do processo de seleção da maior correlação.

Após a última comparação ao final do módulo, onde o maior valor de correlação encontrado é comparado com o valor de limiar, os seguintes sinais são enviados ao filtro Cifi: o sinal “PixelCandidato” que possui apenas um bit e indica se o pixel tratado é ou não candidato do primeiro grau, em caso afirmativo, fornece o seu fator de escala; e o sinal de 16 bits com o próprio coeficiente máximo de correlação também é enviado ao sistema de simulação.

Para o filtro Rafi, é mantida a mesma concepção de comparação em *pipelines*. No entanto, deve-se fazer a seleção entre 36 correlações (pois são usados 36 raios no processamento do filtro Rafi). A figura 59 representa este diagrama que apresentará então um número maior de estágios de *pipeline* (ou registradores) em comparação com o filtro anterior. As informações a serem passadas para o próximo filtro (o filtro Tefi, de análise invariante ao contraste e brilho) passam a ser então o valor da escala mais adequada da rotação, e a indicação através do sinal binário “PixelCandidato” que informa se o pixel processado é um pixel candidato do segundo grau.



**figura 59** Árvore de seleção de correlações para o filtro Rafi.

## 4.2.2. SOLUÇÃO PARA O ACESSO A MEMÓRIA

Assumimos para esta dissertação que a imagem  $A$  de resolução 640x480, onde é feita a busca pela imagem de máscara  $Q$ , é armazenada dentro da própria FPGA. Esta abordagem evita que seja necessário o acesso às memórias externas.

O acesso às memórias externas exige que o sistema aguarde vários ciclos de relógio (dependendo da infra-estrutura física das memórias em uma eventual placa eletrônica de processamento) e fique algumas vezes mais lento, pois seria necessário então aguardar que uma coluna inteira de dados fosse lida da imagem  $A$  antes de se iniciar todo o processo em *hardware*. Para um sistema físico ideal em que se tem uma matriz CWP de maior resolução, seria necessário então ler 53 pixels de 8 bits ao mesmo tempo. Estas questões de infra-estrutura eletrônica são discutidas mais adiante no tópico apropriado de Discussões.

Em uma FPGA de tamanho adequado, em que a imagem é armazenada dentro do próprio dispositivo, é possível, não só receber os 53 pixels de uma única vez, como também acessar esta quantidade de dados em apenas um ciclo de *relógio*. A FPGA da Altera escolhida é um modelo Stratix III com 16Mb de memória.

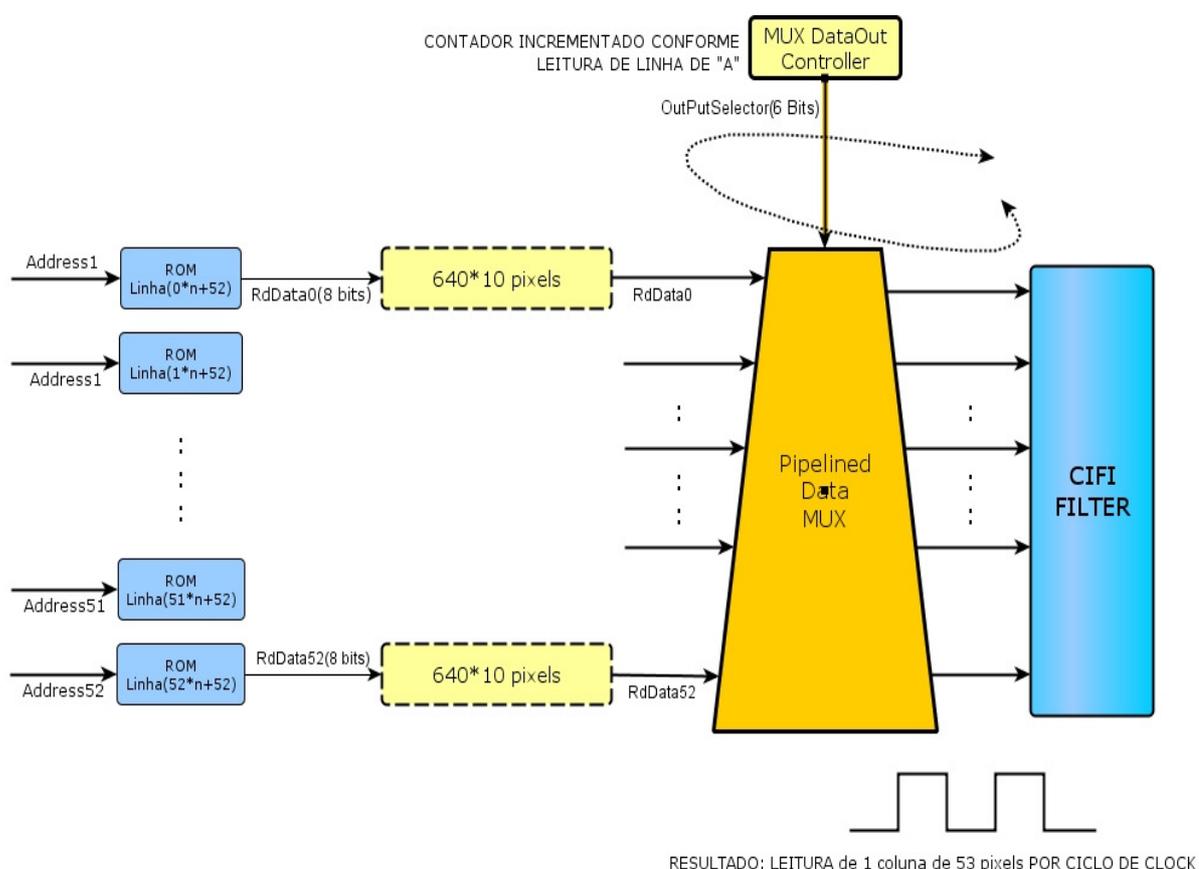
Entretanto, como a janela de processamento CWP percorre a imagem  $A$  em deslocamentos de um pixel, é necessário projetar um *hardware* de forma eficiente para a leitura adequada dos pixels ou a varredura da janela sobre a imagem  $A$  como discutido no tópico 4.2.1.1.1.

A figura 60 apresenta a concepção do *hardware* que permite a leitura de colunas completas, com a resolução da matriz CWP, dos pixels da imagem  $A$  armazenada dentro da FPGA. Este módulo foi projetado de forma tal a permitir a leitura de cada coluna (53 pixels para o pior caso) em apenas um ciclo de *relógio*.

Primeiramente, ao invés de haver uma única memória do tipo ROM, deve-se ter  $L_{CWP}$  unidades de memórias ROM, onde  $L_{CWP}$  é o número de linhas em CWP. Assim, para uma matriz CWP de 53x53 pixels, serão necessárias 53 memórias ROM. Cada endereço da memória possui o mesmo tamanho em bits dos pixels, ou seja 8 bits. A largura destas memórias será igual a  $L_{CWP}$  multiplicado pelo resultado inteiro

da relação do número de linhas entre  $A$  ( $L_A$ ) e  $Q$ , ou seja,  $\frac{L_A}{L_{CWP}}$ .

Na figura 60 ainda é possível identificar a configuração de memórias ROM para a imagem *A* de 640x480 e a imagem *Q* de 53x53. Seriam então necessárias 3 memórias ROM com uma largura de 6400 (640x10) unidades de 8 bits e 50 memória ROM com 5760 (640x9) unidades de 8 bits. Estes valores representam as larguras das memórias. No entanto, a fim de simplificação, toma-se todas as 53 ROMs internas com a mesma largura, de 6400 pixels.



**figura 60** Diagrama em *Hardware* para o acesso aos dados da memória interna em um ciclo de *relógio*.

Os pixels da imagem *A* devem estar armazenados da seguinte forma nas memórias ROM, para uma imagem de máscara com resolução de 53x53: a ROM 0 irá armazenar os 640 pixels da linha 0 da imagem *A*, seguida pelos pixels das linhas 52, 154, 206 e assim por diante. A figura 61 apresenta um exemplo de uma imagem *A* com resolução de 640x480.



**figura 61** Exemplo para uma imagem *A* com resolução de 640x480.

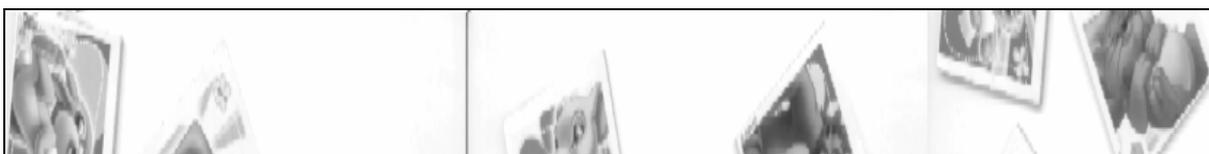
figura 62 representa a mesma imagem *A* convertida para o formato que é armazenado nas memórias internas das da FPGA: uma imagem com resolução de 6400x53. A figura 63 apresenta uma parte em detalhe da mesma imagem da figura anterior.

Para que os pixels de cada linha sejam adequadamente transferidos aos módulos de somatórias do filtro Cifi, os dados lidos das memórias ROM em paralelo devem passar por um multiplexador em *pipeline* que irá selecionar a ordenação adequada das linhas. Esta ordenação irá garantir que a janela de processamento CWP, ao “varrer” a imagem *A*, seja deslocada um pixel para baixo ao final de cada linha da imagem *A*.

Essa ordenação no multiplexador é controlada por um módulo com um contador simples que conta de 0 a 52, com incremento de 1 a cada final de linha da imagem A (ou seja, a cada 640 pixels lidos ou 640 colunas fornecidas ao módulo CWP).



**figura 62** Imagem com resolução de 6400x53 que representa o conteúdo das memórias ROM armazenadas dentro da FPGA.



**figura 63** Fragmento em detalhe, respectivo à seleção indicada na figura anterior, da imagem que representa a imagem A armazenada dentro da FPGA .

### 4.3. ESTRATÉGIA PARA A IMPLEMENTAÇÃO

Como mencionado, optamos pela estratégia de usar uma linguagem em alto nível (linguagem C) para gerar códigos compiláveis em VHDL pelos motivos principais:

- Os códigos em VHDL para este projeto se tornaram muito extensos;
- Grande flexibilidade para entrada de parâmetros diferentes e geração dos códigos correspondentes a estes parâmetros;
- Tarefa de cálculos trigonométricos (que são feitos apenas uma vez, na determinação das coordenadas) são deixadas a cargo da linguagem de alto nível;
- Código em VHDL passa a ser responsável somente pelos cálculos que agregam valor ao processamento;
- Códigos de simulação, análise e testes são gerados com maior facilidade e correspondem aos parâmetros de entrada especificados.

#### 4.3.1. PROGRAMA EM C COMO GERADOR AUTOMÁTICO DE *HARDWARE*

As coordenadas dos pixels, sobre as quais serão calculadas as médias, são automaticamente geradas pelo gerador de *hardware* VHDL que também verifica se há coordenadas repetidas, as quais são descartadas e não entram no processo de somatórias. Com tal abordagem, não é necessário utilizar nenhuma função trigonométrica implementável em *hardware* como em [70,71] nem sistemas com tabelas do tipo *look-up tables* (como em [16]) a fim de calcular as coordenadas dos círculos. Como consequência, enquanto ferramentas trigonométricas em *hardware* do tipo *COrdinate Rotation Digital Computer* (CORDIC) [70, 71] requerem algumas iterações (ciclos de *relógio*) para calcular cada resultado, nossa implementação requer apenas um ciclo de *relógio* por resultado. As soluções por *look-up table* também requerem muito mais recursos de memória a medida em que cresce a resolução das imagens e seria impraticável para as imagens com resolução de

640x480 ou maior. Neste trabalho, as imagens também não sofrem nenhum tipo de simplificação ao longo de todo o processamento.

Para este trabalho foram projetados os diagramas de blocos do *hardware* para os filtros Cifi e Rafi mas as implementações foram feitas somente para o filtro Cifi. Por implementações deve-se entender como o projeto, a programação, análise e verificação de todos os programas que:

- 1) Geram todos os módulos em VHDL compiláveis;
- 2) Todos os códigos em VHDL usados somente nas simulações dos módulos individuais e também aqueles que englobam todo o sistema e lê e escreve em arquivos tipo texto;
- 3) Programas que convertem imagens para texto, criando a infra-estrutura para a simulação com imagens e dados reais;
- 4) Programa que calcula as médias, correlações e cálculos intermediários, tanto em ponto flutuante como em ponto-fixo a fim de se comparar e validar os valores obtidos na simulação (em *hardware*) com aqueles obtidos em *software* (linguagem C).

### 4.3.1.1. PROGRAMAS DESENVOLVIDOS

De maneira geral, as tarefas concluídas para viabilizar o sistema de geração automática de códigos VHDL de processamento e de simulação foram as seguintes:

- Coordenadas dos Círculos em *hardware*.
  - Análise de coordenadas repetidas ou redundantes.
  - Plotagem gráfica dos círculos para análise dos padrões gerados.
- Cálculo do número de estágios de *Pipelines*.
- Constantes para o cálculo das médias por multiplicação em *hardware*.
- Módulo CWP completo, parametrizável e compilável.
- Geração da Matriz  $C_Q$  a partir de máscaras de diferentes resoluções.
  - Cálculo das médias dos círculos.
- Conversão de imagem  $A$  em arquivo tipo texto.
  - Padrão de leitura para o módulo CWP.
  - Simulação de leitura da memória (código VHDL específico).
- Geração de arquivo de controle de:
  - Leitura da imagem  $A$ .
  - Escrita na matriz CWP.
  - *Shifts* de colunas na matriz CWP.
  - Sincronismo de todo processamento (sinais de *Valido*).
- Programas de verificação por *software*:
  - Do valor das médias.
    - Em ponto flutuante e em ponto fixo.
  - Do valor das correlações.
    - Ponto flutuante e em ponto fixo.
- Geração dos módulos de correlações através de entrada de parâmetros:
  - Número de círculos.
- Geração do módulo que seleciona a maior correlação e classifica pixels
  - Em *pipeline*.
  - Com base no valor de limiar.
  - Pixel candidato e escala correspondente.
- Geração de módulo de escrita em arquivo.
- Geração dos módulos de simulação para todas as etapas intermediárias.

- *TestBench* janela de somatória.
- *TestBench* com Somatória e Médias.
- *TestBench* para correlação individual.
- *TestBench* para todas correlações.
- *TestBench* para Média com todas as correlações.
- *TestBench* para Cifi Completo (Leitura/Escreta em Arquivo, Médias, Correlações, Classificação Candidato por Hardware e escrita do resultado em arquivo).
- *TestBench* Cifi Completo com imagem A em 55x53.
- *TestBench* Cifi Completo com imagem A em 160x120.
- *TestBench* Cifi Completo com imagem A em 640x480.

A entrada e configuração dos parâmetros nos programas geradores de código VHDL é feita por linhas de comando. A figura 64 demonstra as opções listadas pelo programa gerador do *hardware* para o módulo de cálculo das médias, enquanto que a figura 65 demonstra as entradas para o programa que gera o *hardware* para o cálculo das correlações.

```
C:\Henrique\PrjCiratefihw\c\code\CifiVHDLGen>cifivhdlgen -h

Usage: cifivhdlgen [-a] [-d number] [-h] [-n number] [-s number] -o output-file -q Template Text File
-a Author Information
-d Square Matrix Dimension (Must be odd)(Min: 25, Max: 53) (dxd) - Default value : 53
-h Help Screen.
-s Space between circles. s>1 and s<d/2 - Default value : 2
-o Output vhd file. Must be Entered
-q Template Image Input Text File

V1.0 cifivhdlgen

This program generate an VHDL output file with a CIFI algorithm implementation.
```

**figura 64** Entrada do programa gerador do módulo CWP de cálculo das médias.

```
Usage: correlvhdlgen [-a] [-d number] [-h] [-n number] [-s number] -o output-file -q Template Text File
-a Author Information
-h Help Screen.
-i Number of input bits per vector value. (Min: 4, Max: 32) - Default value : 8
-n Number of INPUT CIRCLES (Min: 3, Max: 40) - Default value : 14
-o Output vhd file. Must be entered
-r Number of bits of result value. (Min: 8, Max: 64) - Default value : 16

V1.0 correlvhdlgen

This program generate an VHDL output file with a Correlation algorithm implementation.
```

**figura 65** Entrada do programa gerador dos módulos de cálculo das correlações .

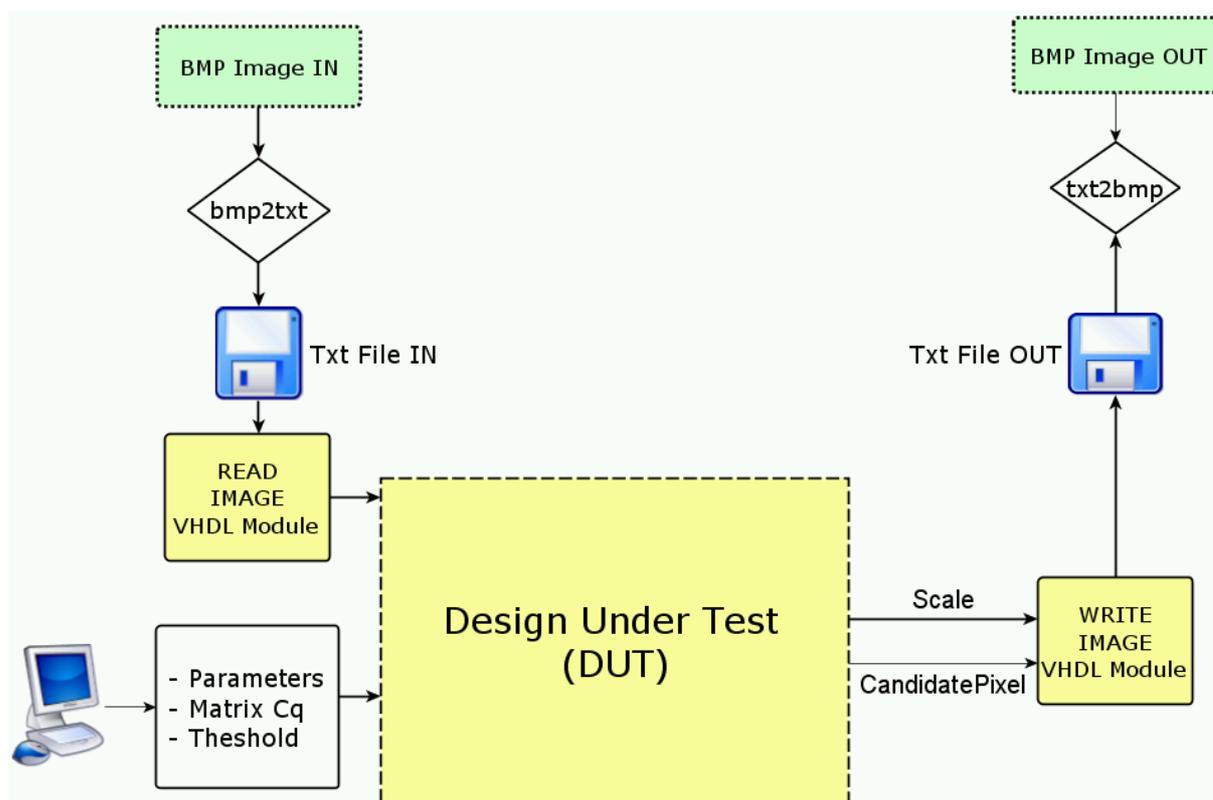
No caso da geração dos códigos para as correlações é preciso gerar o código VHDL para cada correlação separadamente com a respectiva configuração. Visando facilitar a geração e execução do programa gerador de códigos, foi criado um arquivo tipo “.bat” com os parâmetros apropriados por linha de comando. No caso da figura 66,  $n$  representa o número de círculos.

```
BatCorrelVhdlFilesGen.bat (C:\Henrique\PrjCiratefiHW\c\code\CorrelVHDLGen\)  
cd C:\Henrique\PrjCiratefiHW\c\code\CorrelVHDLGen.  
correlvhdlgen -n 5 -o correlacao_Circles05.vhd.  
correlvhdlgen -n 7 -o correlacao_Circles07.vhd.  
correlvhdlgen -n 8 -o correlacao_Circles08.vhd.  
correlvhdlgen -n 9 -o correlacao_Circles09.vhd.  
correlvhdlgen -n 10 -o correlacao_Circles10.vhd.  
correlvhdlgen -n 12 -o correlacao_Circles12.vhd.  
correlvhdlgen -n 14 -o correlacao_Circles14.vhd.
```

**figura 66** Arquivo ".bat" para geração dos códigos VHDL para cada correlação.

## 4.4. ESTRATÉGIA PARA AS SIMULAÇÕES

A figura 67 apresenta um diagrama com a infra-estrutura criada para simular o sistema em *hardware* projetado utilizando imagens reais.



**figura 67** Arquitetura para a infra-estrutura de simulação envolvendo as entradas dos dados da imagem e saída dos resultados.

Principalmente na primeira etapa deste trabalho, em que foram definidas estratégias e soluções para obter-se o melhor desempenho com taxas elevadas de frequências de operação, foram feitas diversas compilações com diferentes dispositivos de modo a verificar a inexistência de erros no código .

Tanto os códigos que utilizam as bibliotecas de manipulação de arquivos quanto os módulos tipo *testbench*, são códigos VHDL não sintetizáveis (não geram *hardware*) e específicos para simulações.

## **CAPÍTULO 5. RESULTADOS**

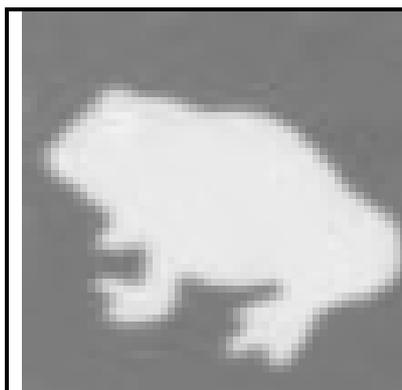
O programa em linguagem em alto nível que implementa o algoritmo Ciratefi foi concebido pelos autores do algoritmo em linguagem C++ [1]. Neste trabalho, para a etapa de validação, análise dos resultados e da precisão, foi preciso programar as etapas de cálculo matemático do algoritmo Ciratefi. Isto foi feito utilizando a linguagem C. O objetivo desta tarefa foi a verificação e validação de todos os resultados matemáticos, não havendo interesse por desempenho em *software* ou eficiência das variáveis utilizadas, estas que eram preocupações dos autores do algoritmo Ciratefi.

Além disso, muitas vezes, estivemos interessados nos valores dos cálculos intermediários e nas diferenças em precisão, por exemplo, para os valores obtidos ainda em *software* para as variáveis em ponto flutuante (*float* ou *double*) e em ponto fixo (*unsigned int*). Esta análise foi fundamental para assegurar a correta implementação em *hardware*, pois os valores obtidos em *software* (com variáveis do tipo inteiro que reproduzissem o mesmo tratamento dado em *hardware*) deveria ser absolutamente o mesmo obtido nas simulações. Para que isto fosse comprovado realizamos os mesmos cálculos do filtro Cifi mas com variáveis em ponto flutuante e em ponto fixo em linguagem C.

Os testes nas etapas finais foram feitos com imagens reais de entrada conforme a estrutura de simulação e análise (figura 67). Para a análise e verificação dos dados foi utilizada uma imagem *A* de entrada de resolução 55X53 pois todos os dados e cálculos intermediários e finais precisavam ser analisados e conferidos.

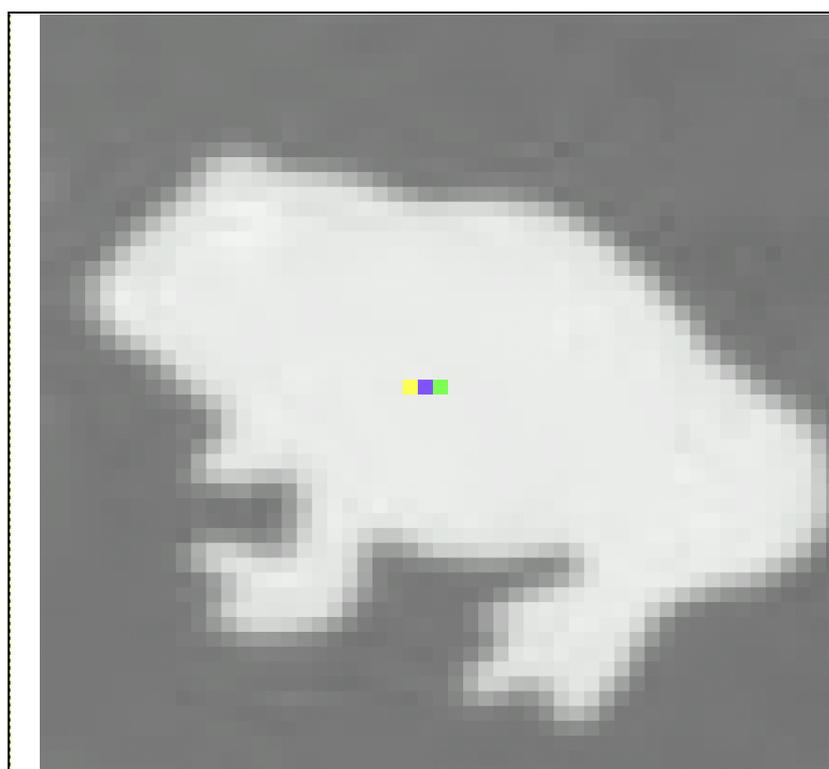
É possível fazer análises e simulações com imagens de maior resolução. Testes iniciais para uma imagem *A* de resolução de 640 x 480 mostraram que o tempo de simulação requer mais de 4 horas de processamento em microcomputador.

Os testes de validação dos cálculos foram feitos utilizando como imagem *A* de 55x53 pixels conforme a figura 68. Por sua vez, a figura 69 expõe em cores diferentes os três pixels processáveis da imagem *A* selecionada, uma vez que as análises foram feitas para o pior caso: janela de processamento CWP com resolução de 53x53 pixels e com a quantidade máxima parametrizada de 14 círculos (contando com o pixel central).



**figura 68** Imagem A de verificação completa dos resultados com resolução de 55x53 pixels.

Para esta análise é portanto, esperado que o valor de correlação máximo ocorra para o terceiro pixel processável (em verde na figura 69).



**figura 69** Mesma imagem A de verificação onde os pixels centrais em cores diferentes apontam os pixels processáveis para esta resolução.

## 5.1. MÓDULOS SINTETIZADOS

Os diagramas em nível de RTL apresentam na forma de circuitos, as ligações entre os módulos. É possível descer em mais baixo nível nos diagramas RTL, de fato, visualizando em circuitos lógicos, como a ferramenta de síntese “converteu” os códigos VHDL.

A figura 70 demonstra o diagrama do filtro Cifi em nível RTL gerado após a síntese dos códigos VHDL no programa Quartus II. Através do diagrama é possível verificar o circuito e as interconexões geradas entre os diferentes módulos de correlações.

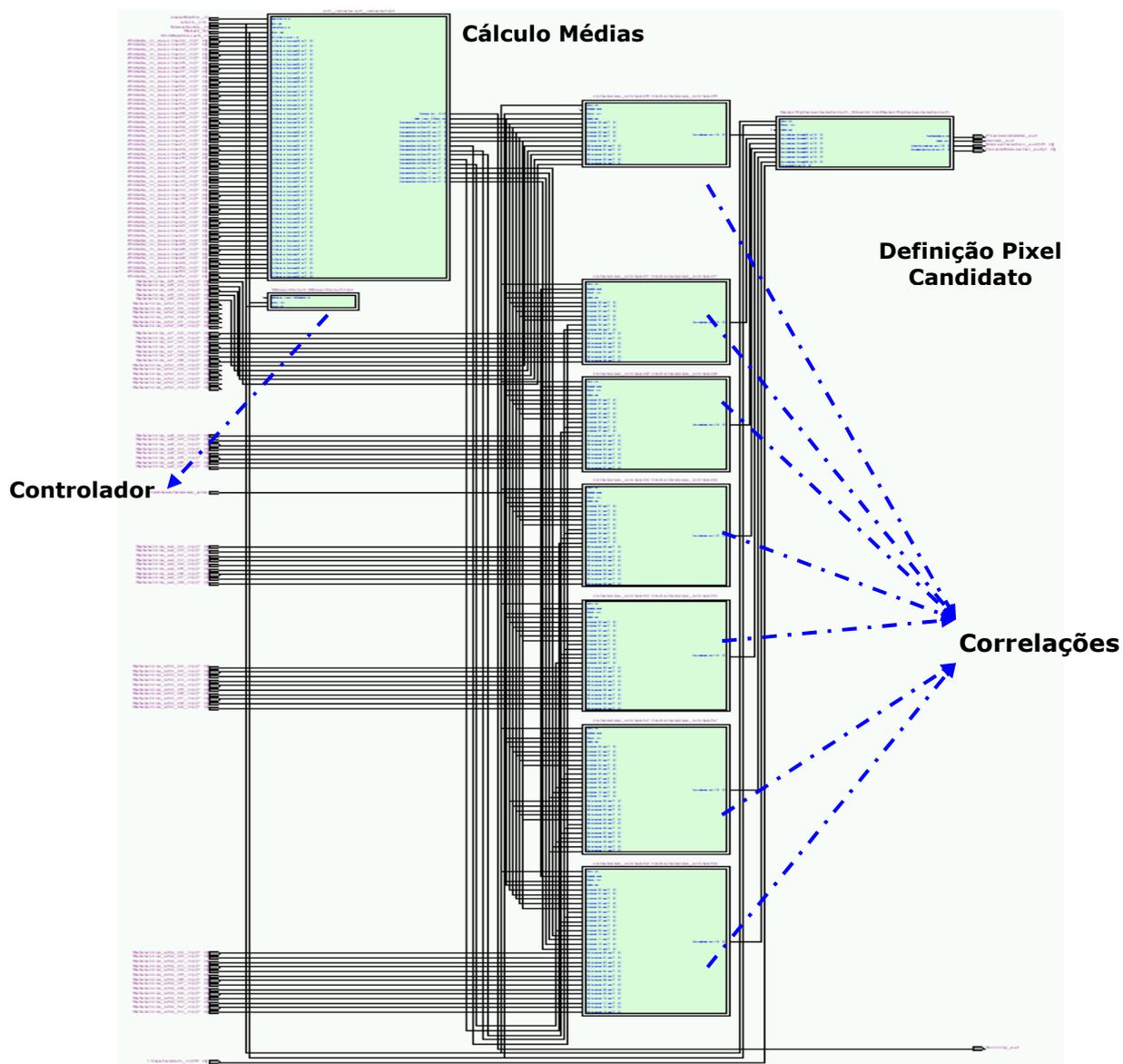
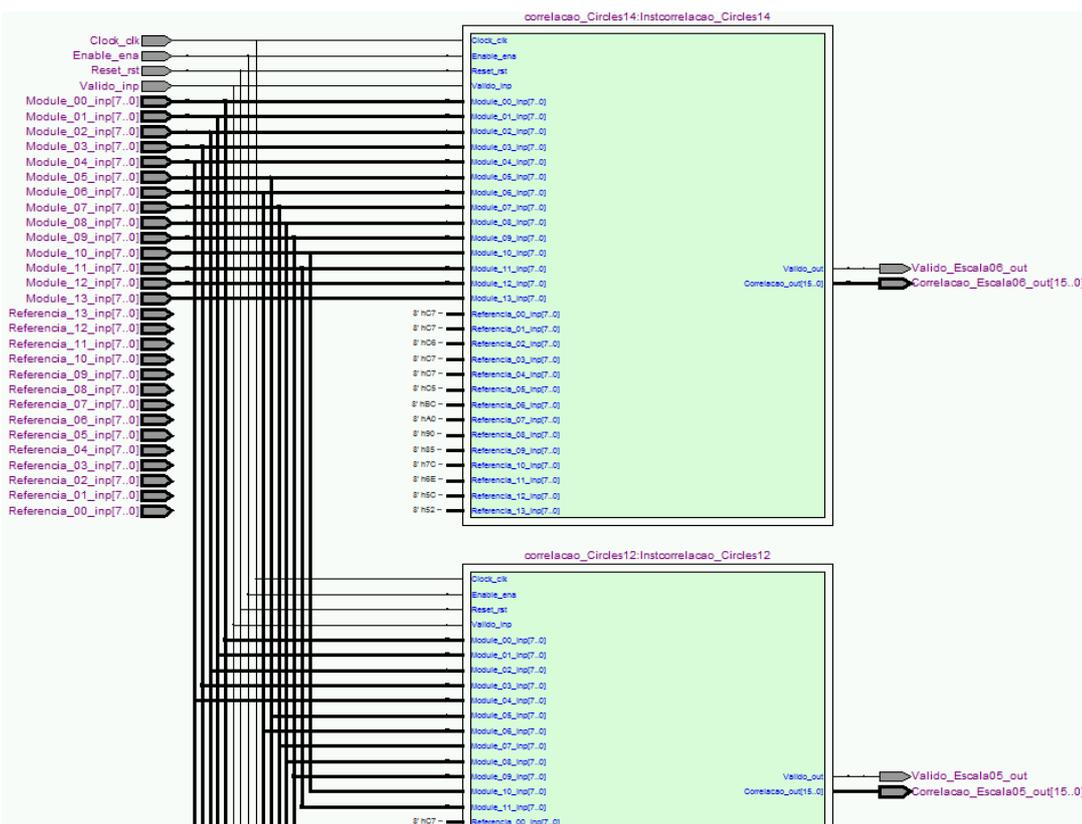


figura 70 Diagrama RTL sintetizado para os módulos de correlação.

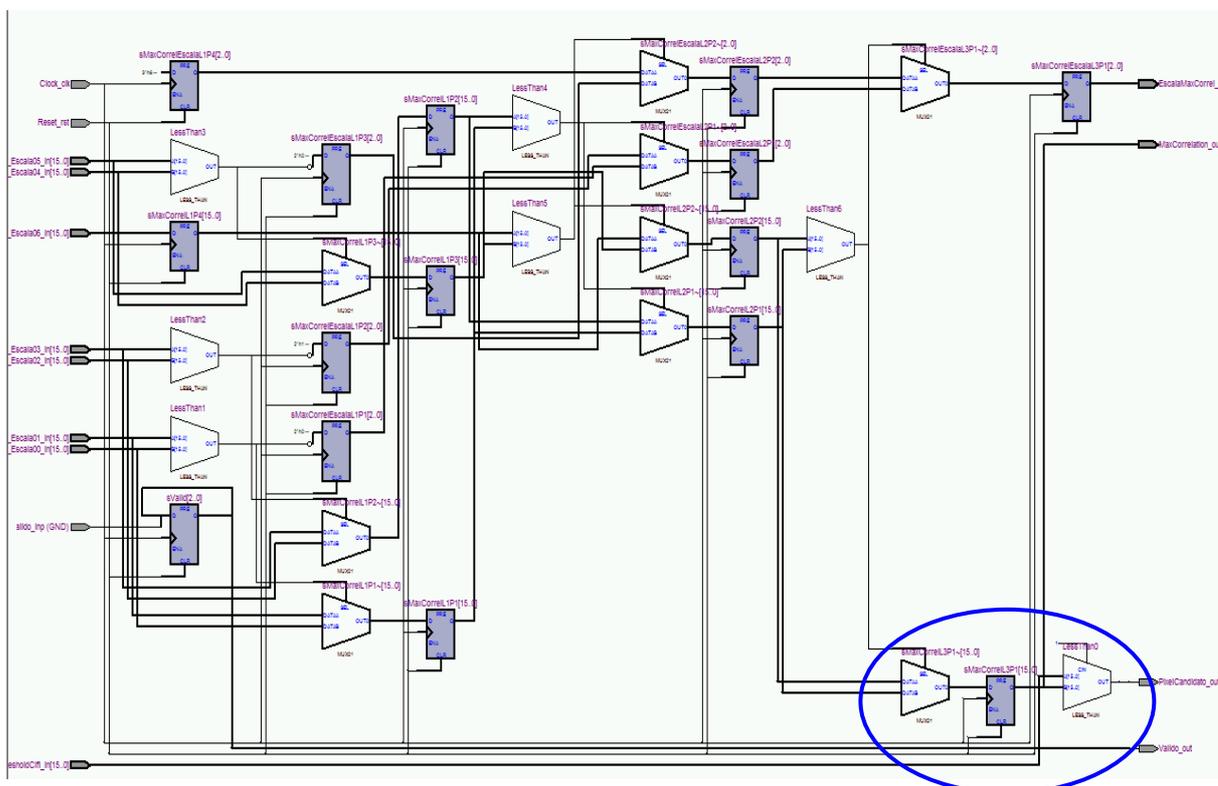
A figura 71 mostra os diagramas RTL em detalhes para os módulos de correlações com 14 e 12 círculos onde é possível verificar os sinais de entrada e saídas, com os resultados das correlações. Estes sinais são entrada para o Módulo de Seleção da Maior Correlação (resultado este que é então, comparado com o valor de limiar para confirmação se o pixel processado será considerado um pixel candidato de primeiro grau).



**figura 71** Diagrama sintetizado em detalhe para os módulos de correlação com 14 e 12 círculos

É possível aprofundar em vários níveis de visualização, os diagramas RTL. No caso da figura 72 por exemplo, está a transcrição em portas lógicas (sendo possível visualizar todos os registradores e multiplexadores) para o Módulo de Seleção da Maior Correlação. Em detalhe, é apresentada a comparação final com o valor de limiar em hexadecimal para verificação se o pixel é ou não candidato (quando o sinal “PixelCandidato” assume valor ‘1’).

Os diagramas RTL em níveis mais baixos para os outros módulos geram diagramas muito mais extensos e complexos, impossibilitando a visualização razoável neste documento.



**figura 72** Diagrama RTL em nível 0 para o Módulo de Seleção da Correlação

Após a síntese de todos os módulos, o programa Quartus II, além de gerar os diagramas de conexões lógicas RTL, também cria diagramas visuais dos módulos gerados, com suas entradas e saídas.

Como exemplo, a figura 73 demonstra a parte inicial e final do diagrama para o módulo topo de hierarquia, filtro Cifi (é um módulo visualmente extenso com muitas entradas). Nesta figura é possível notar também as saídas que indicam o valor da correlação máxima encontrada, a maior escala e se o pixel é ou não um pixel candidato. Como entrada, há os sinais de dados dos pixels correspondentes à coluna CWP, o valor de limiar e os valores referências da matriz  $C_Q$ , entre outros.



**figura 73** Parte inicial e final do diagrama de bloco para o módulo topo de hierarquia, filtro Cifi

A figura 74 por sua vez, demonstra o diagrama de blocos para o módulo integrado de somatórias e cálculo das médias. É possível notar, por exemplo, entre as saídas, os valores de médias calculados (valores de 8 bits).

O módulo topo de hierarquia ou o módulo de *testbench* no caso das simulações, interliga todos os sub-módulos (conectando todas as entradas e saídas). No caso do *testbench* na simulação, o módulo final não possui entradas ou saídas pois ele mesmo gera os estímulos.

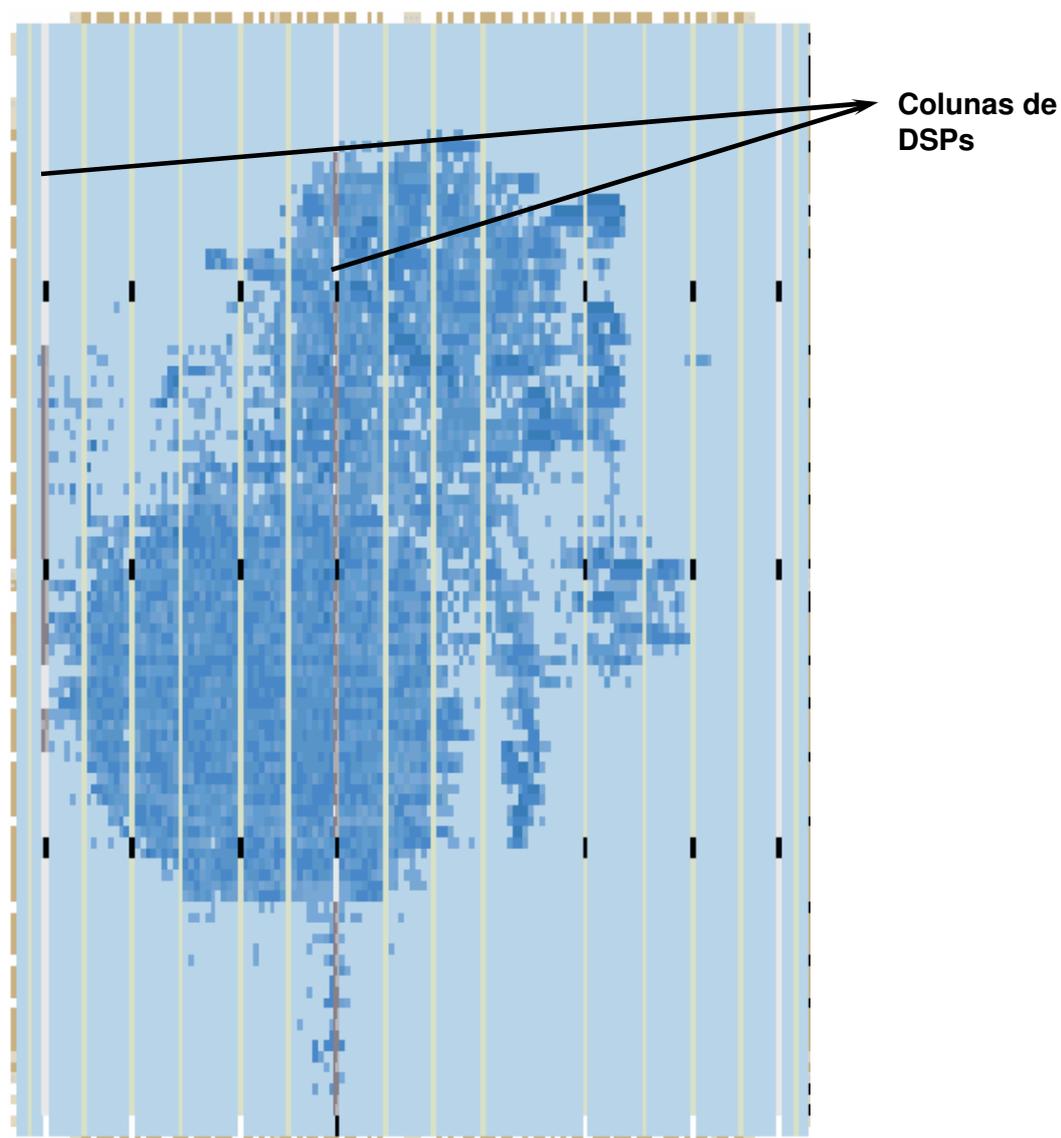
No caso do topo de hierarquia da figura 73 as entradas e saídas serão ligadas aos pinos da FPGA ou conectadas em outros módulos apropriados, como por exemplo, em uma CPU instanciada na própria FPGA para comunicação com o computador e entrada dos parâmetros adequados (valores de referência, de limiar

em ponto-fixo ou outras constantes calculadas em alto nível, como no caso de uma das melhorias futuras propostas mais adiante no item 6.2.1.)



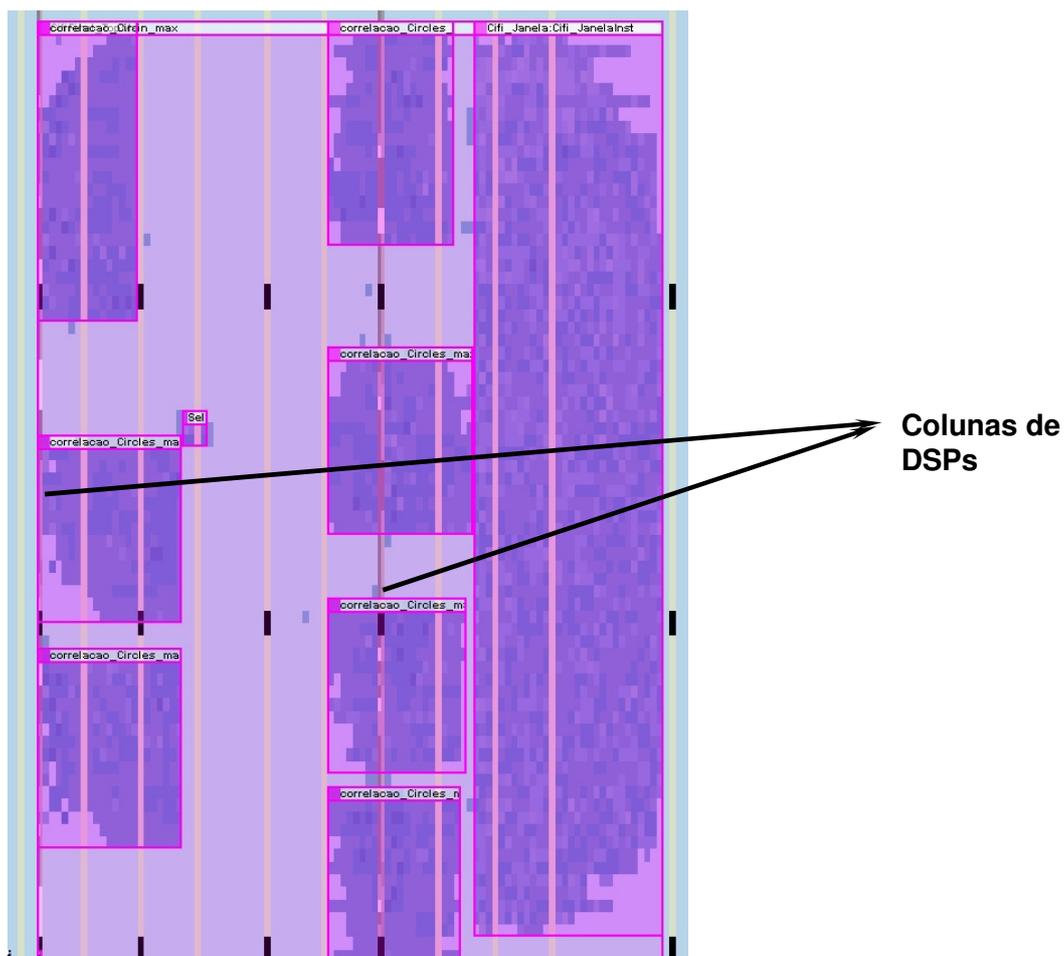
**figura 74** Parte do diagrama de blocos sintetizado para o módulo de Somatória e Cálculo das Médias.

O processo de roteamento do *hardware* sintetizado na FPGA é um processo complexo e pode inclusive requerer cerca de uma hora de compilação em um computador dual core com 8 GB de RAM e 3,2GHz. Ferramentas da Altera como compilação incremental podem tornar a tarefa mais gerenciável pois é possível selecionar módulos individuais para serem roteados. O resultado deste roteamento, além do arquivo de configuração da FPGA, *bitstream*, pode ser também visualizado graficamente.



**figura 75** Leiaute com o módulo Cifi fisicamente roteado na FPGA selecionada.

A figura 75 demonstra este diagrama de roteamento. As regiões mais escuras demonstram maior densidade lógica de recursos utilizados. Nas colunas é possível identificar os blocos físicos de memórias internas e os multiplicadores DSPs que fazem parte da estrutura física do próprio dispositivo FPGA.



**figura 76** Leiaute com o módulo Cifi completo, fisicamente roteado na FPGA utilizando a ferramenta de particionamento.

A figura 76 demonstra o mesmo diagrama para outra abordagem de roteamento, utilizando a ferramenta de particionamento. O resultado permite separar fisicamente os principais módulos dentro da própria FPGA, com o objetivo de otimizar o sistema como um todo.

## 5.2. CÁLCULOS PARA AS SOMATÓRIAS

O *software* que gera os códigos em C cria também arquivos tipo texto com diversas informações sobre os cálculos realizados durante todo o processo de geração dos códigos VHDL. Estas informações foram importantes para o processo de verificação, depuração e correção dos erros encontrados nos cálculos em *hardware*.

A figura 77 demonstra por exemplo, durante a análise das coordenadas dos círculos geradas, o gráfico dos círculos traçados para a maior resolução da matriz de processamento.

Pode-se ainda verificar outros cálculos realizados, como o número máximo de termos a serem somados. Estas análises foram fundamentais para os estudos e dimensionamento dos sinais digitais a serem implementados em *hardware*.

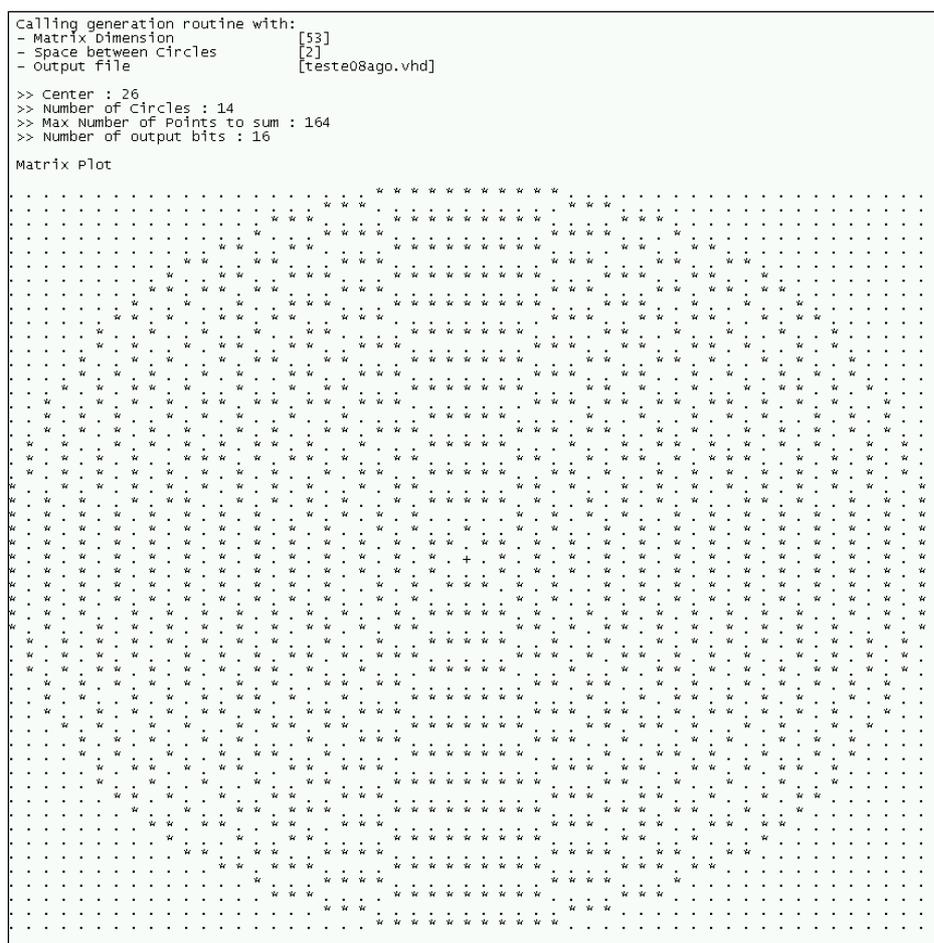


figura 77 Saída gráfica gerada automaticamente para análise de coordenadas para o Módulo de Cálculo das Somatórias e Médias.

A figura 78 explicita outro tipo de análise feita pelo programa desenvolvido em C, ainda antes do início da geração dos códigos em VHDL e trata especificamente das coordenadas dos pontos dos círculos. A maioria dos círculos apresenta pontos repetidos e esta identificação impede que estas coordenadas sejam somadas em *hardware*, de forma redundante.

```

Circle 3 => Number of points calculated: 38   Real number of points: 36
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 32 32 32 31 31 30 29 28 27 26 25 24 23 22 21 21 20 20 20 20 20 21 21 22 23 24 25 26 27 28 29 30 31 31 32 32
Y : 26 27 28 29 30 30 31 31 32 32 32 31 31 30 29 28 27 26 25 24 23 22 22 21 21 20 20 20 21 21 22 22 23 24 25
Total number of Pipelines Levels for Circle 3: 6

Circle 4 => Number of points calculated: 51   Real number of points: 48
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 34 34 34 33 33 33 32 31 30 29 28 27 26 25 24 23 22 21 20 20 19 19 18 18 18 18 19 19 20 20 21 22 23 24 25 26 27 2
Y : 26 27 28 29 30 31 31 32 33 34 34 34 34 34 34 33 33 32 32 31 30 29 28 27 26 25 24 23 22 21 20 20 19 19 18 18 18 1
Total number of Pipelines Levels for Circle 4: 6

Circle 5 => Number of points calculated: 63   Real number of points: 58
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 36 36 36 36 35 35 34 33 32 31 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 17 17 16 16 16 16 16 17 17 17 18 19 20 2
Y : 26 27 28 29 30 31 32 33 34 34 35 35 36 36 36 36 35 35 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 1
Total number of Pipelines Levels for Circle 5: 6

Circle 6 => Number of points calculated: 76   Real number of points: 76
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 38 38 38 38 37 37 37 36 35 35 34 33 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 19 18 17 17 16 15 15 15 14 14 1
Y : 26 27 28 29 30 31 32 33 34 34 35 35 36 37 37 37 38 38 38 38 38 38 38 37 37 37 36 35 35 34 33 33 32 31 30 29 28 27 2
Total number of Pipelines Levels for Circle 6: 7

Circle 7 => Number of points calculated: 88   Real number of points: 88
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 40 40 40 40 39 39 39 38 38 37 37 36 35 34 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 18 17 16 15 15 14 14 1
Y : 26 27 28 29 30 31 32 33 34 34 35 35 36 37 37 38 38 39 39 39 40 40 40 40 40 40 39 39 39 38 38 37 37 36 35 34 34 33 3
Total number of Pipelines Levels for Circle 7: 7

Circle 8 => Number of points calculated: 101  Real number of points: 98
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 42 42 42 42 42 41 41 41 40 39 38 38 37 36 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 19 18 17 16 15 15 1
Y : 26 27 28 29 30 31 32 33 34 35 35 36 37 38 38 39 40 40 41 41 41 42 42 42 42 42 42 42 42 41 41 41 40 40 39 39 38 37 3
Total number of Pipelines Levels for Circle 8: 7

Circle 9 => Number of points calculated: 114  Real number of points: 112
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 44 44 44 44 44 43 43 43 42 42 41 41 40 40 39 38 37 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 1
Y : 26 27 28 29 30 31 32 33 34 35 35 36 37 38 39 40 41 41 42 42 42 43 43 43 44 44 44 44 44 44 43 43 43 42 42 42 4
Total number of Pipelines Levels for Circle 9: 7

Circle 10 => Number of points calculated: 126 Real number of points: 120
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 46 46 46 46 46 45 45 45 44 44 44 43 43 42 41 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 1
Y : 26 27 28 29 30 31 32 33 34 35 36 36 37 38 39 40 41 42 43 43 44 44 45 45 45 45 46 46 46 46 46 46 46 45 45 45 45 4
Total number of Pipelines Levels for Circle 10: 7

Circle 11 => Number of points calculated: 139 Real number of points: 132
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 48 48 48 48 48 47 47 47 46 46 45 45 44 44 43 42 41 40 39 38 37 36 35 34 34 33 32 31 30 29 28 27 26 25 24 23 22 2
Y : 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 44 45 45 46 46 47 47 47 47 48 48 48 48 48 48 48 48 48 4
Total number of Pipelines Levels for Circle 11: 8

Circle 12 => Number of points calculated: 151 Real number of points: 142
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 50 50 50 50 50 49 49 49 48 48 48 47 47 46 45 45 44 43 42 41 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 2
Y : 26 27 28 29 30 31 32 33 34 35 36 37 37 38 39 40 41 42 43 44 44 45 46 46 47 47 48 48 49 49 49 50 50 50 50 50 50 5
Total number of Pipelines Levels for Circle 12: 8

Circle 13 => Number of points calculated: 164 Real number of points: 156
Imprimindo Coordenadas APENAS dos pontos Úteis
X : 52 52 52 52 52 51 51 51 50 50 50 49 48 48 47 47 46 45 45 44 43 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 2
Y : 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 43 44 45 45 46 47 47 48 48 49 50 50 50 51 51 51 52 52 52 5
Total number of Pipelines Levels for Circle 13: 8

Total number of discarded points: 48

```

**figura 78** Saída com os pontos para coordenadas que serão transferidos ao programa em VHDL para processamento.

Para o caso de maior resolução da matriz CWP e com o maior número de círculos (tendo como espaço entre círculos parametrizado como 1 pixel) presentes na figura 78, tem-se ao final da análise a detecção de um total de 48 pontos repetidos e então descartados. Portanto, somente são passados ao gerador de *hardware* as coordenadas para os pontos "úteis".

### 5.3. CÁLCULOS DAS MÉDIAS

Após a geração dos códigos em VHDL para o módulo CWP que calcula as médias dos pontos para cada círculo, iniciou-se a etapa de verificação dos resultados obtidos. Com a infra-estrutura de análise especificada, simulamos a leitura, o preenchimento, o deslocamento, a soma e os cálculos das médias para o módulo CWP.

Calculamos as médias dos círculos em C para diversos pixels, efetuando dois tipos de cálculos: em ponto flutuante e em ponto fixo. A figura 79 demonstra a saída em arquivo texto dos resultados obtidos em *software* para o primeiro dos três pixels processáveis da imagem A da figura 68.

```

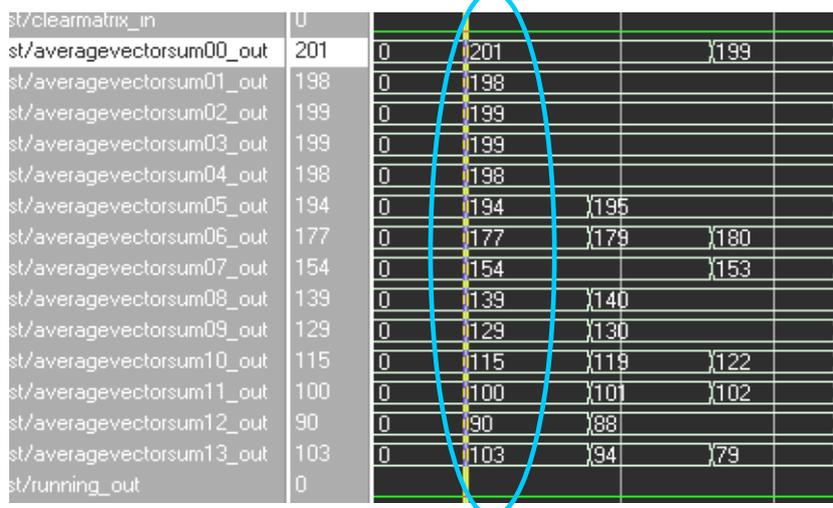
1101 -----
1102 MEDIAS IMAGEM A : Imprimindo dados das medias refrentes a IMAGEM A .
1103 -----
1104 Media do Circulo 0 da Imagem A = 201.0000 e em PF = 201 Hexa PF = C9.
1105 Media do Circulo 1 da Imagem A = 198.8333 e em PF = 198 Hexa PF = C6.
1106 Media do Circulo 2 da Imagem A = 199.1667 e em PF = 199 Hexa PF = C7.
1107 Media do Circulo 3 da Imagem A = 199.0556 e em PF = 199 Hexa PF = C7.
1108 Media do Circulo 4 da Imagem A = 198.6458 e em PF = 198 Hexa PF = C6.
1109 Media do Circulo 5 da Imagem A = 194.3621 e em PF = 194 Hexa PF = C2.
1110 Media do Circulo 6 da Imagem A = 177.5263 e em PF = 177 Hexa PF = B1.
1111 Media do Circulo 7 da Imagem A = 154.3977 e em PF = 154 Hexa PF = 9A.
1112 Media do Circulo 8 da Imagem A = 139.5918 e em PF = 139 Hexa PF = 8B.
1113 Media do Circulo 9 da Imagem A = 129.7232 e em PF = 129 Hexa PF = 81.
1114 Media do Circulo 10 da Imagem A = 115.9833 e em PF = 115 Hexa PF = 73.
1115 Media do Circulo 11 da Imagem A = 100.2879 e em PF = 100 Hexa PF = 64.
1116 Media do Circulo 12 da Imagem A = 90.8803 e em PF = 90 Hexa PF = 5A.
1117 Media do Circulo 13 da Imagem A = 103.2436 e em PF = 103 Hexa PF = 67.

```

**figura 79** Resultados dos cálculos (em *software*) das médias para a imagem A da figura 69 cujo pixel processado é o (26,26) com cálculos em ponto flutuante e também com cálculos feitos como no *hardware*.

Em seguida, os resultados obtidos em *software* foram comparados com os resultados obtidos em *hardware* através do programa de simulação ModelSIM. Estes resultados estão expostos na figura 80 e podem ser comparados com os resultados em *software* da figura anterior.

É importante salientar que todos os módulos em VHDL para manipulação de arquivos e para estímulos e simulação (*testbenchs*) são gerados automaticamente, de acordo com as parametrizações de entrada selecionadas no programa em C de geração de código VHDL.



st/clearmatrix_in	0				
st/averagevectorsum00_out	201	0	201	199	
st/averagevectorsum01_out	198	0	198		
st/averagevectorsum02_out	199	0	199		
st/averagevectorsum03_out	199	0	199		
st/averagevectorsum04_out	198	0	198		
st/averagevectorsum05_out	194	0	194	195	
st/averagevectorsum06_out	177	0	177	179	180
st/averagevectorsum07_out	154	0	154		153
st/averagevectorsum08_out	139	0	139	140	
st/averagevectorsum09_out	129	0	129	130	
st/averagevectorsum10_out	115	0	115	119	122
st/averagevectorsum11_out	100	0	100	101	102
st/averagevectorsum12_out	90	0	90	88	
st/averagevectorsum13_out	103	0	103	94	79
st/running_out	0				

**figura 80** Resultados dos cálculos das médias para a Imagem A da figura 69 realizados em *hardware*. É possível compará-los com resultados apresentados na figura anterior.

A figura 81 expõe trecho do código VHDL gerado automaticamente pelo programa em C projetado. Nesta parte do código, as médias dos círculos relativos à Janela de Processamento que percorre a imagem A são calculados através da multiplicação por constantes. Estas constantes representam o inverso em ponto fixo (com os 16 bits referentes à parte fracionária) da quantidade útil de pixels presentes em cada círculo.

Para o caso em questão, o sistema foi parametrizado com 14 círculos em uma matriz de processamento de 53x53 pixels. Estes parâmetros referem-se às configurações para o pior caso.

```

1922 -- Calculo das Médias das Somas. Realização da Multiplicação.
1923 -----
1924 sResultMultCirc01 <= (sVectorSumCirc01 * x"1555"); -- Multiplicacao da Soma dos Pixels do Circulo 1 (# Pixels = 12) pela constante em Pto Fixo (0.16) = 5461.
1925 sResultMultCirc02 <= (sVectorSumCirc02 * x"0AAA"); -- Multiplicacao da Soma dos Pixels do Circulo 2 (# Pixels = 24) pela constante em Pto Fixo (0.16) = 2730.
1926 sResultMultCirc03 <= (sVectorSumCirc03 * x"071C"); -- Multiplicacao da Soma dos Pixels do Circulo 3 (# Pixels = 36) pela constante em Pto Fixo (0.16) = 1820.
1927 sResultMultCirc04 <= (sVectorSumCirc04 * x"0555"); -- Multiplicacao da Soma dos Pixels do Circulo 4 (# Pixels = 48) pela constante em Pto Fixo (0.16) = 1365.
1928 sResultMultCirc05 <= (sVectorSumCirc05 * x"0469"); -- Multiplicacao da Soma dos Pixels do Circulo 5 (# Pixels = 58) pela constante em Pto Fixo (0.16) = 1129.
1929 sResultMultCirc06 <= (sVectorSumCirc06 * x"035E"); -- Multiplicacao da Soma dos Pixels do Circulo 6 (# Pixels = 76) pela constante em Pto Fixo (0.16) = 862.
1930 sResultMultCirc07 <= (sVectorSumCirc07 * x"02E8"); -- Multiplicacao da Soma dos Pixels do Circulo 7 (# Pixels = 88) pela constante em Pto Fixo (0.16) = 744.
1931 sResultMultCirc08 <= (sVectorSumCirc08 * x"029C"); -- Multiplicacao da Soma dos Pixels do Circulo 8 (# Pixels = 98) pela constante em Pto Fixo (0.16) = 668.
1932 sResultMultCirc09 <= (sVectorSumCirc09 * x"0249"); -- Multiplicacao da Soma dos Pixels do Circulo 9 (# Pixels = 112) pela constante em Pto Fixo (0.16) = 585.
1933 sResultMultCirc10 <= (sVectorSumCirc10 * x"0222"); -- Multiplicacao da Soma dos Pixels do Circulo 10 (# Pixels = 120) pela constante em Pto Fixo (0.16) = 546.
1934 sResultMultCirc11 <= (sVectorSumCirc11 * x"01F0"); -- Multiplicacao da Soma dos Pixels do Circulo 11 (# Pixels = 132) pela constante em Pto Fixo (0.16) = 496.
1935 sResultMultCirc12 <= (sVectorSumCirc12 * x"01CD"); -- Multiplicacao da Soma dos Pixels do Circulo 12 (# Pixels = 142) pela constante em Pto Fixo (0.16) = 461.
1936 sResultMultCirc13 <= (sVectorSumCirc13 * x"01A4"); -- Multiplicacao da Soma dos Pixels do Circulo 13 (# Pixels = 156) pela constante em Pto Fixo (0.16) = 420.

```

**figura 81** Parte do código em VHDL responsável pelo cálculo de divisão através da multiplicação pelo inverso do número de pixels para cada círculo.

Para o cálculo destas constantes foram feitos os seguintes procedimentos: tomando-se o número de pontos úteis para cada círculo (exposto como comentário ainda no código da mesma figura) calculou-se o seu inverso, e do resultado,

multiplicou-se por 65536 a fim de obter o valor da constante em ponto fixo com 16 bits para a parte fracionária.

Neste cálculo das médias estamos multiplicando um sinal de 16 bits (sinal *sVectorSumCirc* que contém a soma dos pixels do referente círculo) por uma constante em hexadecimal também de 16 bits. O resultado desta multiplicação é um sinal com 32 bits.

Como estamos calculando o valor médio dos pixels contidos nos círculos, o valor esperado deve possuir 8 bits (imagens estão em tons de cinza de 8 bits). O valor da média desejado está contido portanto, no intervalo do bit 23 ao 16. Isto porque a constante multiplicada em ponto fixo é do tipo "0.16" (nenhum bit para a parte inteira e 16 bits para a parte fracionária), ou seja, desprezamos os 16 bits menos significativos do resultado obtido de 32 bits da multiplicação. Como seguramente, os valores não podem ter mais do que 8 bits, não é necessário verificações de saturação entre os bits 32 e 24 do resultado da multiplicação.

## 5.4. CÁLCULOS DAS CORRELAÇÕES

Com os cálculos das médias sendo realizados corretamente em *hardware* pode-se então analisar os cálculos das correlações visto que o Módulo de Cálculo das Correlações recebe como entrada os dados das médias calculados.

Os valores das médias dos círculos que são entrada para este módulo referem-se à imagem *A* (onde a Janela de Processamento “varre” os pixels) e também às diferentes escalas da imagem de máscara.

O pré-processamento das escalas das imagens de máscara, conforme o algoritmo do Ciratefi, gera uma matriz que contém o valor das médias por círculo e por imagem escalonada. A figura 82 demonstra a matriz  $C_Q$  calculada pelo programa em C desenvolvido neste trabalho.

Após o cálculo da matriz  $C_Q$ , o programa em C gera os códigos VHDL de cálculo da correlação, ligando estes valores das médias com as entradas apropriadas dos módulos em *hardware* de cálculo da correlação.

```

IMPRIMINDO MATRIZ CQ .
-----
                CIRCULOS.
                0      1      2      3      4      5      6      7      8      9      10      11      12      13.
-----
Escala 0 -> 199(C7) | 198(C6) | 171(AB) | 128(80) | 84(54) | .
Escala 1 -> 199(C7) | 199(C7) | 197(C5) | 172(AC) | 137(89) | 109(6D) | 82(52) | .
Escala 2 -> 199(C7) | 198(C6) | 198(C6) | 193(C1) | 157(9D) | 133(85) | 116(74) | 88(58) | .
Escala 3 -> 199(C7) | 198(C6) | 199(C7) | 197(C5) | 176(B0) | 145(91) | 131(83) | 109(6D) | 86(56) | .
Escala 4 -> 199(C7) | 198(C6) | 199(C7) | 198(C6) | 189(BD) | 158(9E) | 139(8B) | 127(7F) | 104(68) | 85(55) | .
Escala 5 -> 199(C7) | 199(C7) | 199(C7) | 199(C7) | 197(C5) | 182(B6) | 153(99) | 139(8B) | 127(7F) | 110(6E) | 90(5A) | 80(50) | .
Escala 6 -> 199(C7) | 198(C6) | 199(C7) | 199(C7) | 198(C6) | 196(C4) | 180(B4) | 153(99) | 140(8C) | 130(82) | 122(7A) | 103(67) | 88(58) | 79(4F)
.
-----
FIM DA IMPRESSÃO DA MATRIZ CQ .

```

**figura 82** Exemplo de uma matriz de processamento  $C_Q$  gerado pelo *software* em C gerador de códigos em VHDL.

A figura 83 expõe trechos dos códigos VHDL gerados para diferentes módulos de correlação onde é possível notar como os valores calculados na matriz  $C_Q$  são passados na forma de constantes em hexadecimal às respectivas entradas destes módulos de correlação. Em elipses, na mesma figura, estão indicadas as constantes ligadas às entradas destes módulos de correlação.

```

1666 -----
1667 -- correlacao_Circles14 Device.
1668 -----
1669 Instcorrelacao_Circles14:
1670 COMPONENT correlacao_Circles14.
1671 -----
1672 -- Port Map .
1673 -----
1674 PORT map(.
1675 -- Common signals.
1676 Reset_rst      => reset_rst,
1677 Enable_ena     => EnableCorrelacao_ena,
1692 Module_13_inp => AverageVectorSum13_out,
1693 Referencia_00_inp => x"C7",
1694 Referencia_01_inp => x"C6",
1695 Referencia_02_inp => x"C7",
1696 Referencia_03_inp => x"C7",
1697 Referencia_04_inp => x"C6",
1698 Referencia_05_inp => x"C4",
1699 Referencia_06_inp => x"B4",
1700 Referencia_07_inp => x"99",
1701 Referencia_08_inp => x"8C",
1702 Referencia_09_inp => x"82",
1703 Referencia_10_inp => x"7A",
1704 Referencia_11_inp => x"67",
1705 Referencia_12_inp => x"58",
1706 Referencia_13_inp => x"4F",
1707 Correlacao_out    => Correlacao_Escala06_out,
1708 Valido_out        => Valido_Escala06_out,
1709 Valido_inp        => Valid_from_CifiSum_out,
1456 -----
1457 -- correlacao_Circles05 Device.
1458 -----
1459 Instcorrelacao_Circles05:
1460 COMPONENT correlacao_Circles05.
1461 -----
1462 -- Port Map .
1463 -----
1464 PORT map(.
1465 -- Common signals.
1466 Reset_rst      => reset_rst,
1467 Enable_ena     => EnableCorrelacao_ena,
1468 Clock_clk      => Clock_clk,
1469 Module_00_inp  => AverageVectorSum00_out,
1470 Module_01_inp  => AverageVectorSum01_out,
1471 Module_02_inp  => AverageVectorSum02_out,
1472 Module_03_inp  => AverageVectorSum03_out,
1473 Module_04_inp  => AverageVectorSum04_out,
1474 Referencia_00_inp => x"C7",
1475 Referencia_01_inp => x"C6",
1476 Referencia_02_inp => x"AB",
1477 Referencia_03_inp => x"80",
1478 Referencia_04_inp => x"54",
1479 Correlacao_out  => Correlacao_Escala00_out,
1480 Valido_out      => Valido_Escala00_out,
1481 Valido_inp      => Valid_from_CifiSum_out
1482 );.
1483 -----
1484 -----
1485 -- correlacao_Circles07 Device.
1486 -----
1487 Instcorrelacao_Circles07:
1488 COMPONENT correlacao_Circles07.

```

figura 83 Arquivo VHDL de topo de hierarquia recebendo os valores das médias pré-processadas na Matriz  $C_Q$ .

Após o cálculo da matriz  $C_Q$  e a correta ligação destes sinais aos módulos de correlação, pode-se iniciar o processo de comparação dos resultados de correlações obtidos no *software* e em *hardware* na simulação.

No programa em C desenvolvido, pode-se selecionar as coordenadas do pixel da imagem  $A$ , em relação ao qual, o *software* calculará as correlações. No *hardware* o processo é contínuo, ou seja, sobre todos os pixel processáveis serão calculadas as correlações na ordem de varredura dos pixel sobre a imagem  $A$ .

O programa em C calcula as correlações para o pixel da imagem  $A$  selecionado, utilizando primeiramente, variáveis com precisão em ponto flutuante. Estes resultados estão expostos na figura 85 em "Valores da correlação com cálculo em ponto flutuante".

```

25 -----
26 VALORES DA CORRELACAO COM CALCULO EM PONTO FLUTUANTE .
27 -----
28 CORRELACAO ESCALA 0 = 0.557654.
29 CORRELACAO ESCALA 1 = 0.817701.
30 CORRELACAO ESCALA 2 = 0.874823.
31 CORRELACAO ESCALA 3 = 0.929856.
32 CORRELACAO ESCALA 4 = 0.962348.
33 CORRELACAO ESCALA 5 = 0.990864.
34 CORRELACAO ESCALA 6 = 0.989278.

```

**figura 84** Resultados dos cálculos das correlações feitas em *software*. Para obtê-los foram usadas variáveis com precisão em ponto flutuante (valores para a coordenada (26,26) da imagem A).

Em seguida, todos os cálculos são refeitos utilizando variáveis do tipo inteiro e reproduzindo a forma como o próprio *hardware* manipula os sinais e resultados, ou seja, empregando a divisão em ponto fixo. Estes resultados em ponto fixo estão expostos na figura 85 em "Imprimindo os resultados dos cálculos das correlações em ponto fixo".

Os resultados em ponto fixo, em hexadecimal devem corresponder totalmente ao obtido no *hardware* e neste caso a divisão é feita deslocando o numerador de 16 bits para a esquerda (multiplicando o numerador por 65536) antes de se dividir o numerador com o deslocamento pelo denominador calculado. O resultado desta divisão está indicando pela elipse em azul ainda na figura 85 onde é indicado também, a título de ilustração, o mesmo resultado convertido para sua parte fracionária. Esta conversão é feita dividindo-se o resultado obtido em ponto fixo, por 65536.

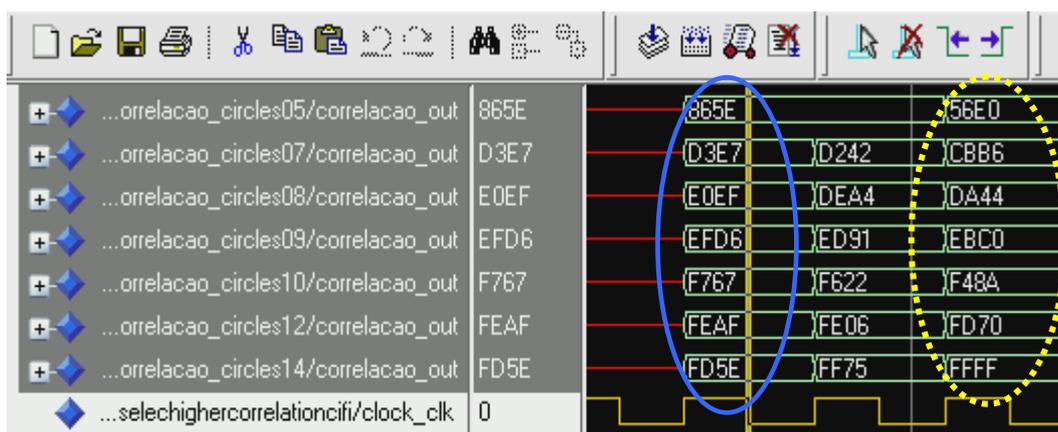
```

Imprimindo os resultados dos Calculos das Correlacoes em Ponto Fixo.
-----
CORRELACAO COM CALCULO EM PONTO FIXO PARA ESCALA 0 = 0.524872 e em Hexa = 865E.
CORRELACAO COM CALCULO EM PONTO FIXO PARA ESCALA 1 = 0.827744 e em Hexa = D3E7.
CORRELACAO COM CALCULO EM PONTO FIXO PARA ESCALA 2 = 0.878647 e em Hexa = E0EF.
CORRELACAO COM CALCULO EM PONTO FIXO PARA ESCALA 3 = 0.936859 e em Hexa = EFD6.
CORRELACAO COM CALCULO EM PONTO FIXO PARA ESCALA 4 = 0.966415 e em Hexa = F767.
CORRELACAO COM CALCULO EM PONTO FIXO PARA ESCALA 5 = 0.994858 e em Hexa = FEAF.
CORRELACAO COM CALCULO EM PONTO FIXO PARA ESCALA 6 = 0.989716 e em Hexa = FD5E.

```

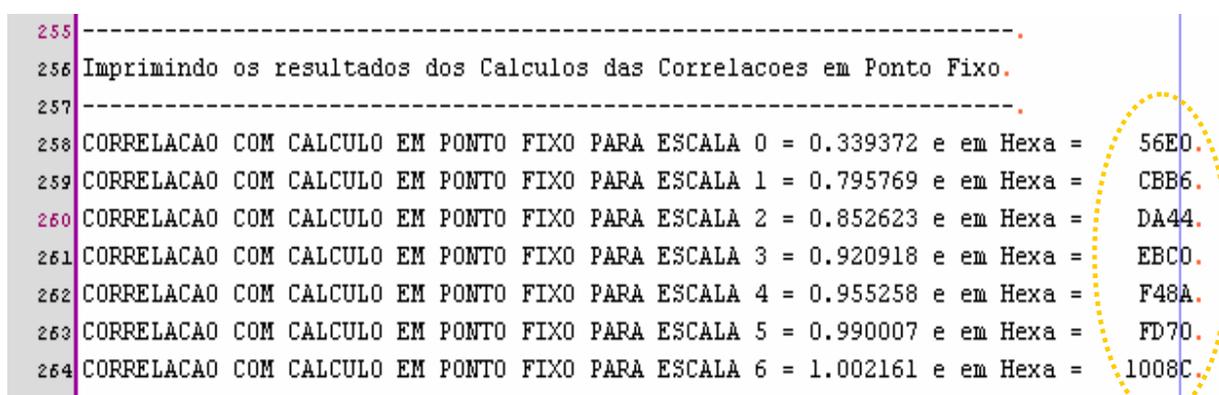
**figura 85** Resultados das correlações com cálculos que reproduzem o processamento feito em *hardware* em ponto fixo (valores para coordenada (26,26)).

Na figura 86 tem-se o resultado da correlação em *hardware* para os 3 pixels processáveis da imagem A utilizada (figura 69) que possui resolução de 55x53 pixels. É possível comparar estes resultados com aqueles obtidos em *software* na figura 85 para o pixel (26,26) e na figura 87 para o pixel (28,26) quando a correlação será máxima para a maior escala. Pode-se notar ainda pela figura 86 que há sete resultados de correlações disponíveis por ciclo de relógio.



**figura 86** Resultados dos cálculos das correlações feitas em *hardware*. É possível verificar em detalhe o sinal de relógio.

Ainda na figura 87 nota-se que para a maior escala a correlação resultou em um valor maior do que 1 devido aos arredondamentos dos cálculos em *hardware*. Para contornar o problema, foi analisado os resultados de correlação deslocando-se o intervalo em *bits* destes resultados finais de modo que não houvesse saturação. Estas questões de precisão são discutidas em maior profundidade no item 6.1 desta dissertação.



**figura 87** Resultados dos cálculos das correlações feitas em *software* que reproduzem cálculo em *hardware* em ponto fixo (valores para coordenada (28,26)).

Na figura 88 estão expostos os resultados em ponto flutuante, em que o pixel processado foi o de coordenada (28,26). Nestes resultados, onde foram usadas variáveis com precisão máxima, nota-se que para a maior escala a correlação máxima foi atingida.

```
----- .  
VALORES DA CORRELACAO COM CALCULO EM PONTO FLUTUANTE .  
----- .  
.  
CORRELACAO ESCALA 0 = 0.315214.  
CORRELACAO ESCALA 1 = 0.772031.  
CORRELACAO ESCALA 2 = 0.844554.  
CORRELACAO ESCALA 3 = 0.914291.  
CORRELACAO ESCALA 4 = 0.952509.  
CORRELACAO ESCALA 5 = 0.986816.  
CORRELACAO ESCALA 6 = 1.000000.  
.
```

**figura 88** Resultados dos cálculos das correlações feitas em *software* para cálculos em ponto flutuante (valores para coordenada (28,26)).

O trabalho de verificação dos valores dos cálculos intermediários foi maior na análise das correlações, quando muitos resultados intermediários foram escritos em arquivo texto para acompanhamento e verificação de todos eles através do simulador ModelSim, que permite acompanhar ao longo do tempo todos estes dados.

Nesta fase de depuração, uma verificação individual de todos os sinais, para todas as 7 correlações foi feita entre *software* e *hardware* até que todos os valores coincidissem e as diferenças encontradas com o cálculo em ponto flutuante fossem compreendidas. Na figura 89 estão expostos os principais resultados intermediários para os cálculos de duas das 7 correlações usadas.

```

159 IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 0 QUE POSSUI 5 CIRCULOS.
160 -----
161 vPF_SomaXiYi [0] - Em PF = 155336 - Em Hexa = 25EC8 .
162 vPF_SomaXi [0] - Em PF = 995 - Em Hexa = 3E3 .
163 vPF_SomaYi [0] - Em PF = 780 - Em Hexa = 30C .
164 vPF_SomaXi2 [0] - Em PF = 198011 - Em Hexa = 3057B .
165 vPF_SomaYi2 [0] - Em PF = 131486 - Em Hexa = 2019E .
166 NumeradorSEMSHIFT[0] - Em PF = 580 - Em Hexa = 244 .
167 Numerador [0] - Em PF = 38010880 - Em Hexa = 2440000 .
168 DenominadorX[0] - Em PF = 5 - Em Hexa = 5 .
169 DenominadorY[0] - Em PF = 221 - Em Hexa = DD .
170 Denominador FINAL[0] - Em PF = 1105 - Em Hexa = 451 .
171 -----
172 CORRELACAO ESCALA 0 EM HEXA = 865E .
173 CORRELACAO ESCALA 0 EM FLOAT = 0.524895 .
174 .
175 -----
176 IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 1 QUE POSSUI 7 CIRCULOS.
177 -----
178 vPF_SomaXiYi [1] - Em PF = 215618 - Em Hexa = 34A42 .
179 vPF_SomaXi [1] - Em PF = 1366 - Em Hexa = 556 .
180 vPF_SomaYi [1] - Em PF = 1095 - Em Hexa = 447 .
181 vPF_SomaXi2 [1] - Em PF = 266976 - Em Hexa = 412E0 .
182 vPF_SomaYi2 [1] - Em PF = 184969 - Em Hexa = 2D289 .
183 NumeradorSEMSHIFT[1] - Em PF = 13556 - Em Hexa = 34F4 .
184 Numerador [1] - Em PF = 888406016 - Em Hexa = 34F40000 .
185 DenominadorX[1] - Em PF = 53 - Em Hexa = 35 .
186 DenominadorY[1] - Em PF = 309 - Em Hexa = 135 .
187 Denominador FINAL[1] - Em PF = 16377 - Em Hexa = 3FF9 .
188 -----
189 CORRELACAO ESCALA 1 EM HEXA = D3E7 .
190 CORRELACAO ESCALA 1 EM FLOAT = 0.827759 .

```

**figura 89** Resultados intermediários dos cálculos das correlações feitas em *software*. Valores expostos apenas para a correlação com 5 e 7 círculos.

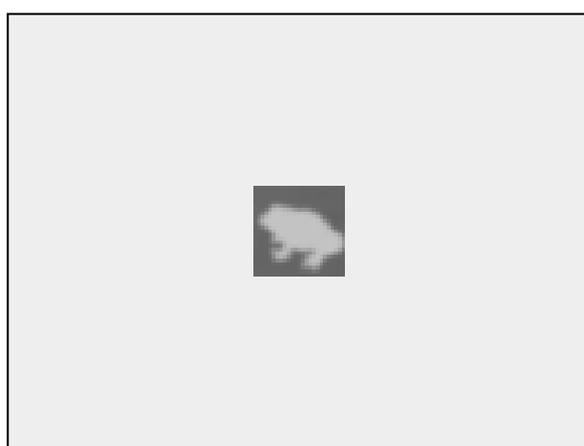
## 5.5. TESTES COM IMAGENS

Uma vez comprovada a acuracidade dos cálculos de correlações para o *hardware* projetado, foi possível concretizar de forma completa o ambiente de simulação proposto no item 4.4 com o diagrama da figura 67. Nesta abordagem, o *hardware* de simulação processa os pixels de qualquer imagem e armazena os resultados (das correlações e escalas mais adequadas) para cada pixel processado.

### 5.5.1. IMAGENS EM BAIXA RESOLUÇÃO

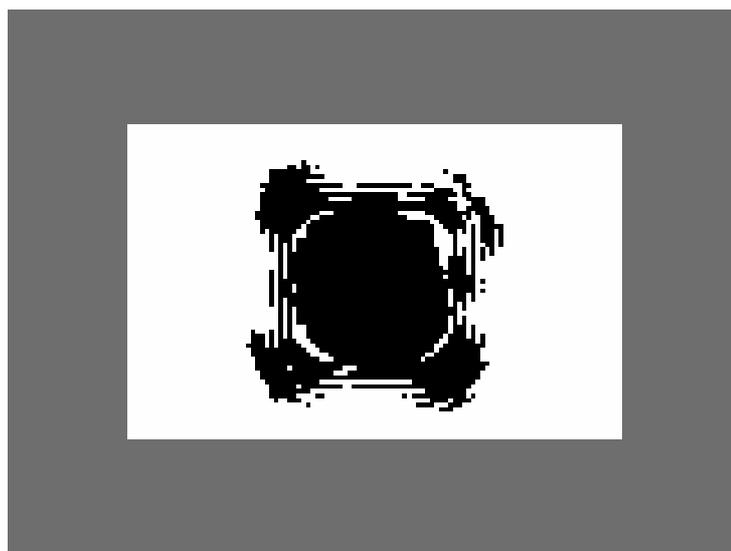
Devido ao tempo de simulação ser bastante considerável e proporcional à resolução da imagem *A* de testes, os primeiros ensaios foram feitos com imagens com resolução de 160x120. O tempo para simulação completa para imagens nesta resolução foi de aproximadamente 20 minutos em computadores de 3Ghz.

Para imagens com resolução de 160x120 e tendo a Janela de Processamento CWP que varre a imagem *A*, resolução de 53x53 pixels, são feitas 7.169 correlações. A figura 90 apresenta a imagem *A* utilizada neste primeiro teste com imagens. Foi colocada apenas uma das 7 imagens de máscara no centro da imagem, sem rotação. A resolução da imagem central era de 25x25 pixels.

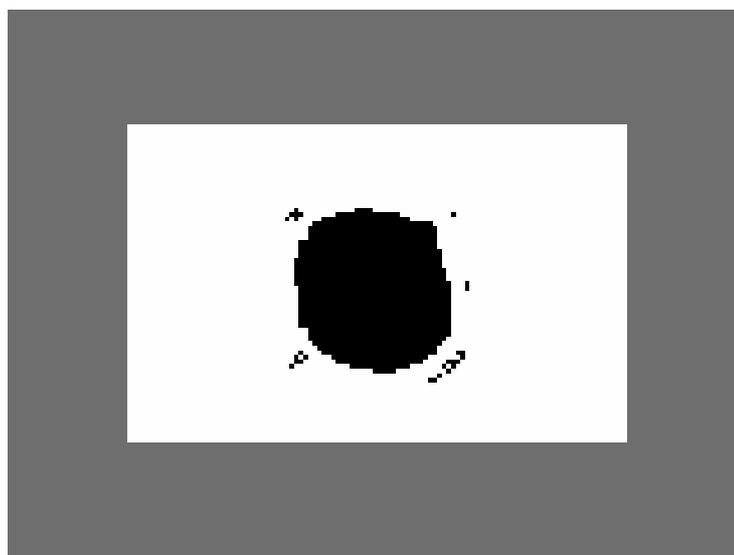


**figura 90** Imagem de teste para o simulador com resolução de 160x120.

As figuras 91 e 92 apresentam os mapas de pixels candidatos selecionados, para os valores de limiares indicados sob cada figura. Os pixels pretos representam os pixels candidatos do primeiro grau. Pode-se notar que a borda acinzentada representa os pixels não processáveis. A espessura desta borda corresponde à metade da resolução da Janela de Processamento CWP, para os casos testados.



**figura 91** Imagem do mapa de pixels candidatos para limiar de 0,90.



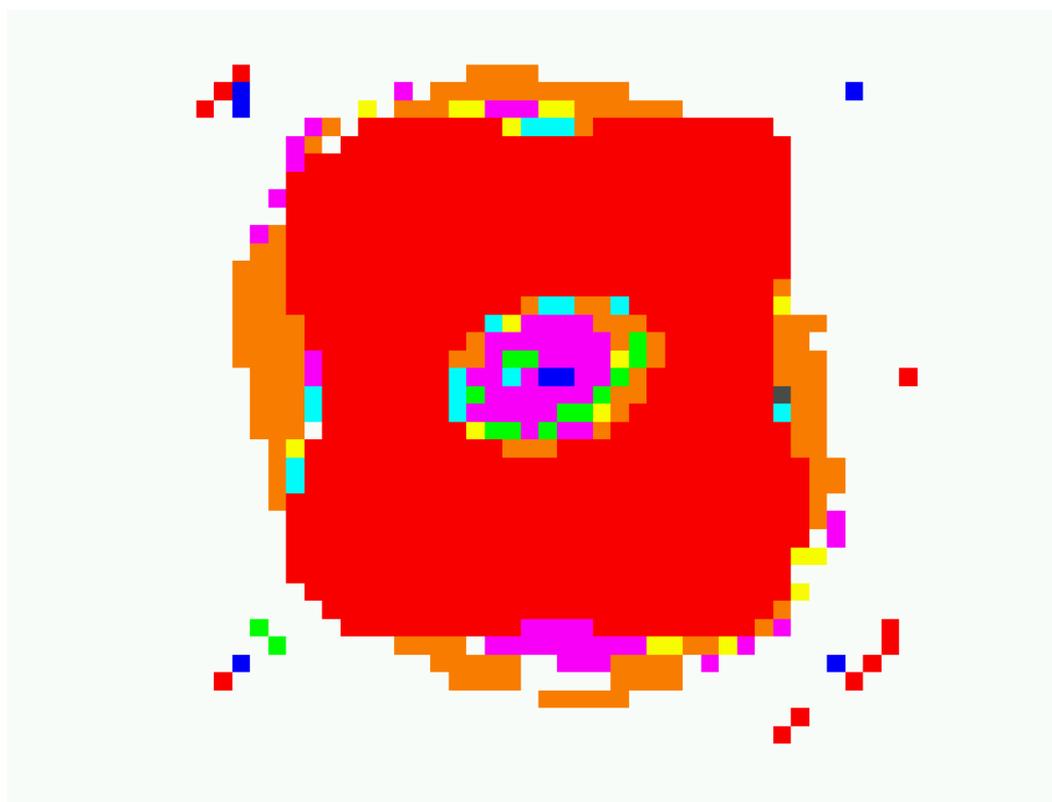
**figura 92** Imagem do mapa de pixels candidatos para limiar de 0,95.

Para cada pixel processado, o *hardware* informa o valor da escala de maior correlação. Esses dados são gravados pelo simulador em um arquivo específico em formato texto. A figura 93 apresenta alguns dos resultados obtidos.

Cifi_EscalaMaxCorreIO	
3533	0006.
3534	0006.
3535	0000.
3536	0000.
3537	0002.
3538	0002.
3539	0003.
3540	0003.
3541	0003.
3542	0004.

**figura 93** Resultado informando escala mais apropriada para cada pixel candidato.

A figura 94 apresenta em detalhe o mapa de pixels candidatos em que cada cor, em cada pixel candidato se refere a uma determinada escala. Esta escala foi a que, para cada pixel, apresentou o maior coeficiente de correlação em relação às demais escalas.



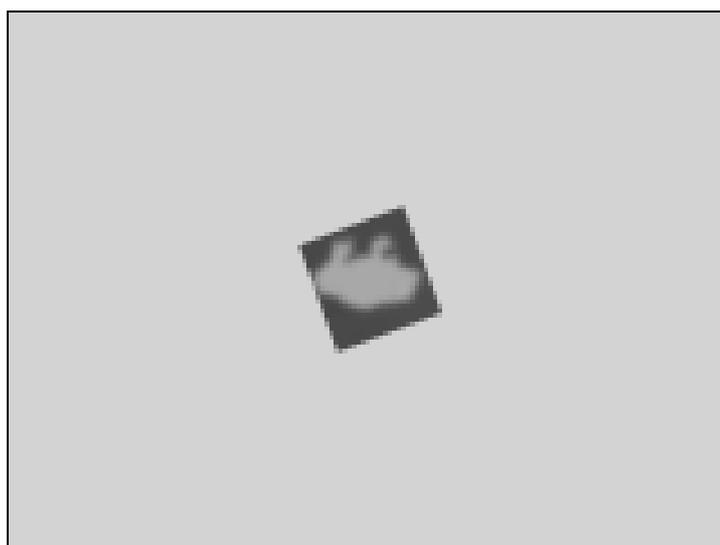
**figura 94** Imagem em detalhe do mapa de pixels candidatos com cores indicando diferentes atribuições de escala.

As configurações de cores referentes à figura 94 e as escalas respectivas são:

- Escala 0 - 17x17 - Verde Claro
- Escala 1 - 25x25 - Azul
- Escala 2 - 31x31 - Vermelho
- Escala 3 - 35x35 - Laranja
- Escala 4 - 39x39 - Amarelo
- Escala 5 - 45x45 - Magenta
- Escala 6 - 53x53 - Verde Escuro

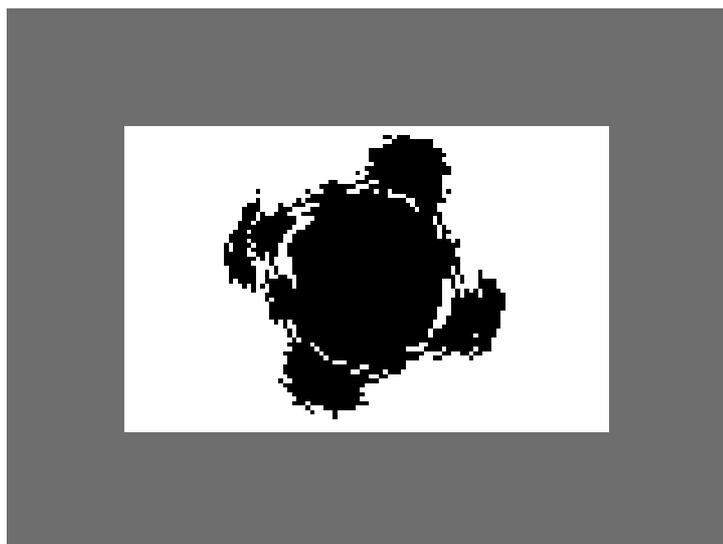
## 5.5.2. IMAGENS EM BAIXA RESOLUÇÃO ROTACIONADAS

Em seguida, foram realizados alguns ensaios, rotacionando-se a imagem do objeto a ser localizado (figura 95).

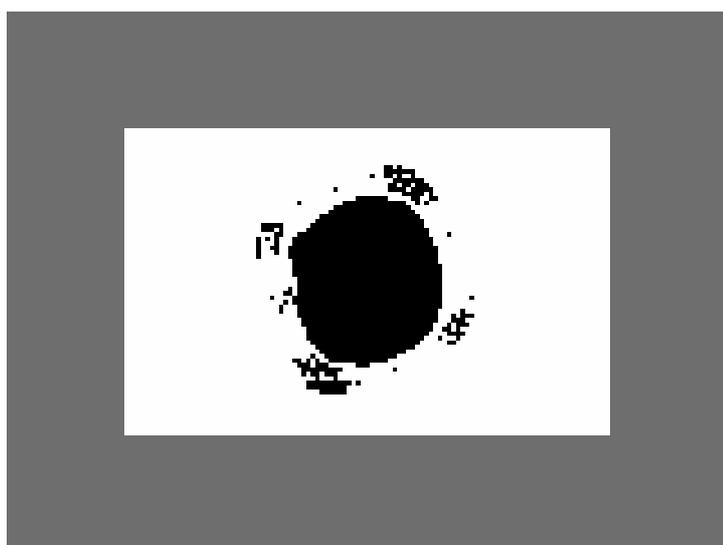


**figura 95** Imagem A de testes com sub-imagem rotacionada.

As próximas imagens (figura 96 e 97) demonstram, para diferentes valores de limiar, os correspondentes mapas de pixels candidatos para a imagem com o objeto a ser localizado.



**figura 96** Imagem do mapa de pixels candidatos para limiar de 0,90.



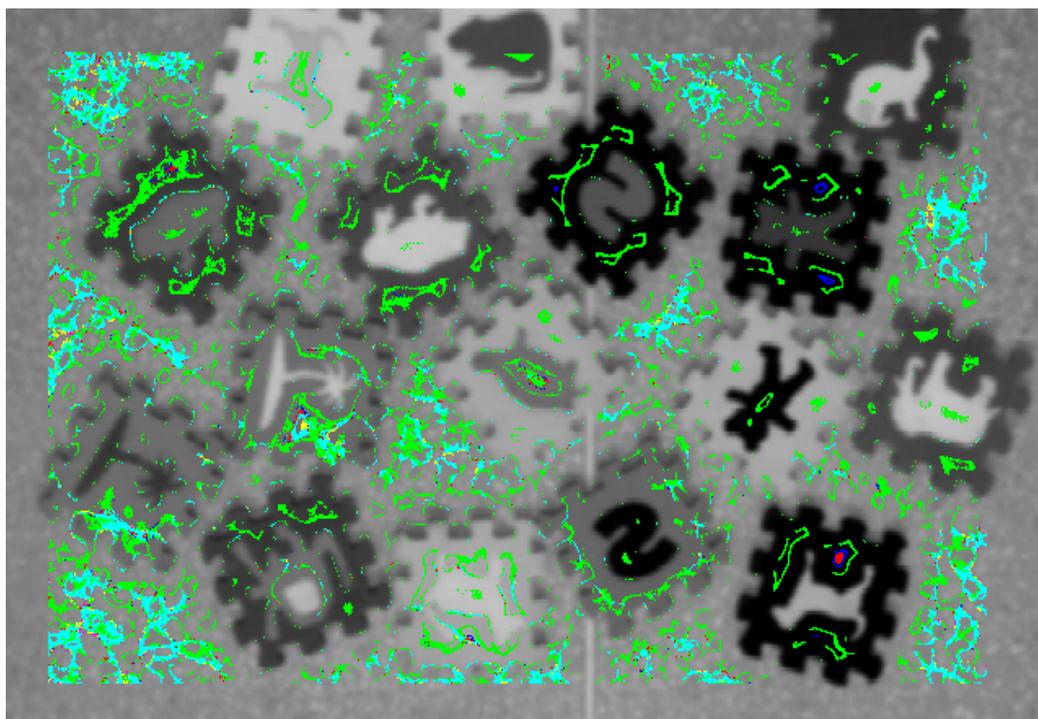
**figura 97** Imagem do mapa de pixels candidatos para limiar de 0,95.

### 5.5.3. IMAGENS EM 640x480

O passo final desta proposta de trabalho, foi simular o *hardware* para imagens de resolução de 640x480 com imagens semelhantes aos trabalhos desenvolvidos em [1]. Este processo de simulação gasta cerca de 4 horas em um computador de 3GHz.

As figura 98, 99 e 100 apresentam os Mapas de Pixels Candidatos para a imagem *A* ensaiada e para valores crescentes do limiar; tendo ainda a imagem do "sapo" como imagem de máscara.

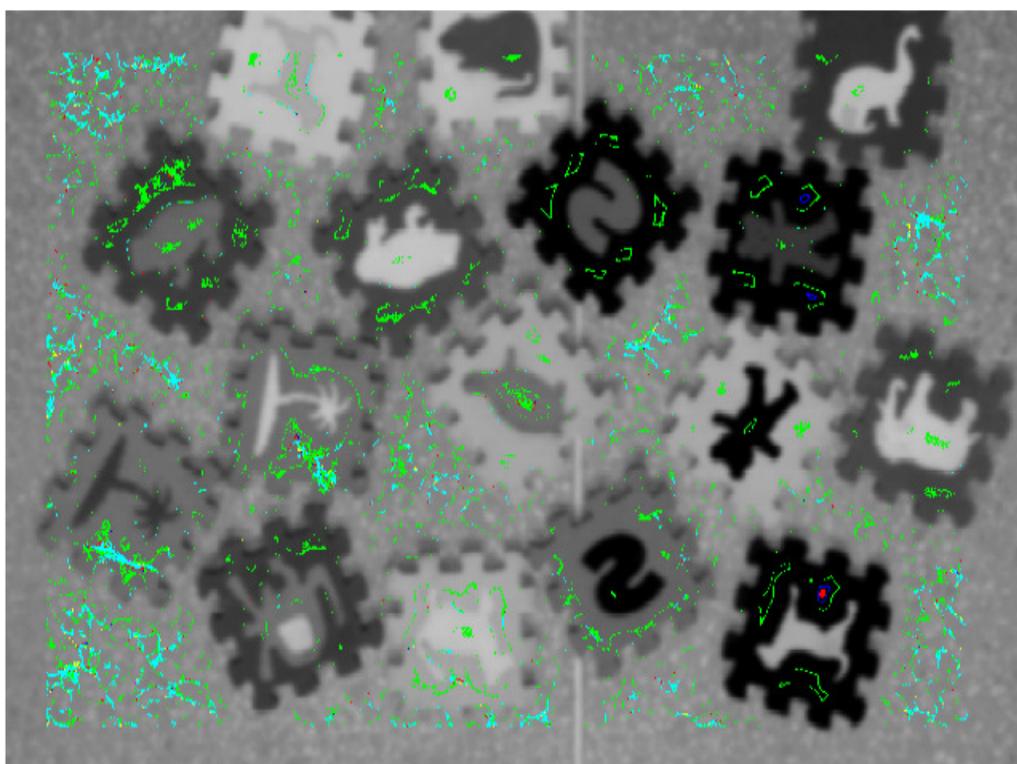
Os pixels coloridos representam diferentes escalas para os pixels candidatos. Nota-se a diminuição da quantidade de pixels candidatos com o aumento do valor de limiar.



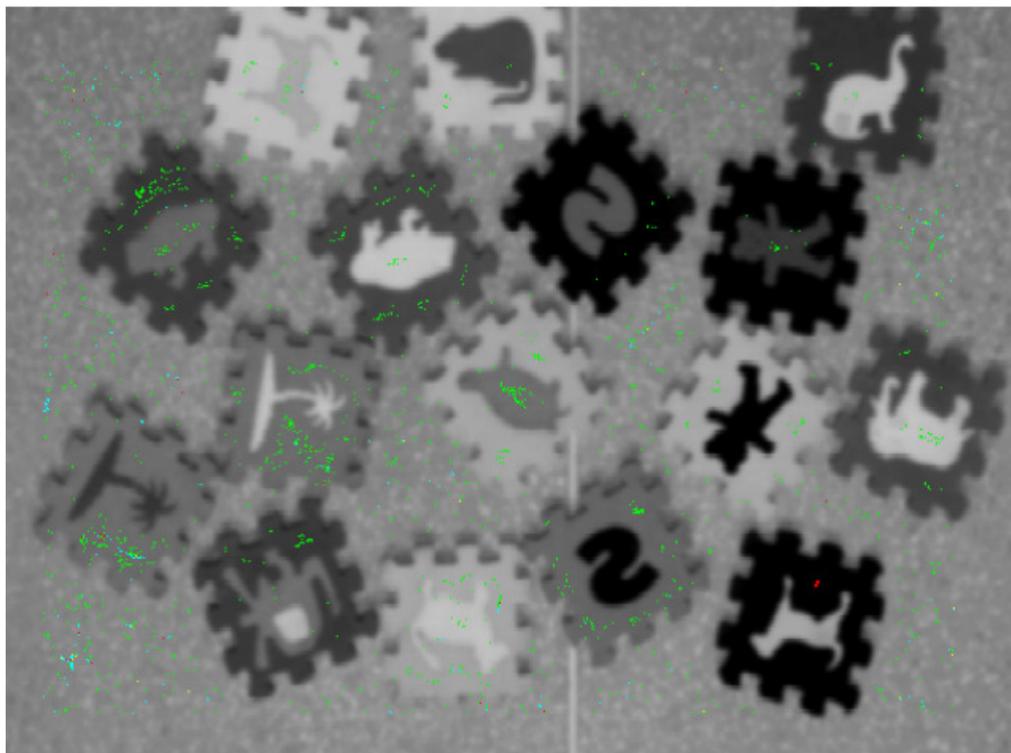
**figura 98** Imagem do mapa de pixels obtido com um valor de limiar baixo.

Para imagens *A* que apresentam diferentes objetos com tons de cinza, notou-se que ocorria saturação do resultado de correlação obtido pois eram tomados os 16 bits menos significativos do resultado final. Por isso, a implementação em *hardware* apresentava uma quantidade excessiva de pixels candidatos.

O sinal em *hardware* com o resultado da correlação pode ter tamanhos que variam desde 38 bits para a correlação da menor escala, até 40 bits para o módulo de correlação da maior escala. No módulo da divisão em *hardware*, o tamanho em bits do resultado da divisão é o mesmo do numerador. Após análises com diferentes imagens, a fim de que não ocorresse mais saturação do resultado em *hardware*, notamos como intervalo de bits mais adequado como sendo do bit 29 ao 14. Tomando estes intervalos, os limiares usados em *hardware* não possuem correspondência direta em *software*, sendo tomados em valores em hexadecimal de 16 bits.



**figura 99** Imagem do mapa de pixels obtido com um valor de limiar intermediário.



**figura 100** Imagem do mapa de pixels obtido com um valor de limiar elevado.

A análise destas imagens demonstra como o fator de limiar pode diminuir consideravelmente o número de pixels candidatos do primeiro grau. Nota-se que os pixels que representam a imagem de máscara buscada continuam a ser localizados. A diminuição do número de pixels candidatos favorece o desempenho do algoritmo como um todo pois menos pixels serão processados pelo próximo filtro, o filtro Rafi.

## 5.6. RESULTADOS FINAIS

Na etapa de análise da melhor estratégia para o desenvolvimento do módulo de cálculo das médias, a primeira questão foi sobre qual o modo mais eficiente de fazer com que a matriz de processamento percorresse os pixels da imagem *A*. Na primeira análise, que foi implementada e testada, a janela de processamento poderia percorrer os pixels não só na direção horizontal, mas também na vertical. A vantagem desta concepção frente ao que foi implementado seria o fato de não ser necessário aguardar a leitura de todas as colunas da matriz (53 colunas para a maior resolução), deixando de haver a mencionada latência de leitura das colunas a cada mudança de linha na varredura da imagem *A* pela janela de processamento.

Ainda nesta primeira etapa, a árvore de processamento implementada possuía muitos menos níveis, em comparação com a implementação em definitivo com somatórias dois-a-dois.

As primeiras análises foram feitas para uma FPGA Stratix III com capacidade de 340.000 elementos lógicos (família EP3S340). Apenas o módulo de somatórias, com a janela de processamento tendo a possibilidade de se deslocar tanto na vertical quanto na horizontal, sintetizou um *hardware* com cerca de 77.000 elementos lógicos, o que é um módulo exageradamente extenso.

Fizemos ainda, análises de freqüências máximas de operação para o módulo de somatórias com apenas um estágio de *pipeline* e com mais estágios de *pipeline* para somas registradas, dois-a-dois em paralelo (conceito de árvore de somadores). O resultado foi que para o primeiro caso, a freqüência máxima de operação ficou em 34Mhz e para o último caso foi de 370Mhz, utilizando uma árvore de somadores e deslocamento da Janela CWP apenas na horizontal. Portanto, no projeto, não foi interessante haver flexibilidade na varredura se não é possível obter maiores freqüências de operação.

## 5.7. DESEMPENHOS

Todas as análises quantificadas neste trabalho foram realizadas para o pior caso, ou seja, para a matriz de processamento com resolução de 53x53 pixels, com 13 círculos mais o ponto central e, finalmente, o tratamento de 7 escalas.

A Tabela 1 mostra as principais características do sistema implementado e analisado, com o número de círculos para cada escala, a resolução e o respectivo fator de escala utilizado.

Escala	Número de Círculos	Resolução (pixels)	Fator de Escala
0	5	17	0,5
1	7	25	0,7
2	8	31	0,9
3	9	35	1,0
4	10	39	1,1
5	12	45	1,3
6	14	53	1,5

**Tabela 1.** Parâmetros utilizados para a validação do sistema.

Nesta seção, nós apresentamos as análises de tempo para cada módulo individualmente com base no dispositivo da Altera Stratix III, EP3SL340H1152C3. Na Tabela 2 é indicado, para o pior caso testado, os recursos utilizados e o desempenho dos módulos CWP de cálculo das somatórias e das médias e o módulo de correlações (onde estão instanciadas as 7 correlações). É importante salientar que tais limites para parâmetros no pior caso foram tomados a partir do trabalho em [1] mas podem eventualmente ser expandidos.

Para o módulo filtro Cifi completo, a frequência máxima obtida ficou abaixo das frequências obtidas para os módulos isoladamente, fator este que era esperado, uma vez que a densidade lógica do sistema completo é maior. A frequência máxima final atingida, foi de 202MHz.

Todas as informações sobre as frequências máximas de cada módulo foram obtidas através do software da Altera QuartusII versão 7.2sp3, com a ferramenta

*Classical Timing Analyser* no modo de análise para o caso mais lento (*Slow Model Analysis*).

Módulo	Frequência Máxima (Mhz)	Tamanho em Lógica	Latência
<b>CWP (Somatórias e Médias)</b>	380	31.258 (12%)	9
<b>Correlações</b>	258	21.538 (1%)	75

Modelo FPGA: EP3SL340H1152C3

**Tabela 2.** Tamanho e recursos lógicos para cada módulo.

A figura 101 apresenta as informações fornecidas após compilação do sistema completo do filtro Cifi em *hardware*. É possível notar as unidades lógicas e de processamento consumidas para geração do sistema. O consumo de recursos lógicos ficou em cerca de 24% do dispositivo FPGA selecionado. Não foram utilizadas ferramentas de otimização que não estivessem disponíveis na versão *WebEdition* (gratuita) do programa QuartusII.

Entity	Combinational AL...	ALMs	Dedicated Logic Registers	DSP Elements	DSP 9x9	DSP 12x12	DSP 18x18	Pins	Combinational with no register ALUT/register pair	Register-Only ALUT/register pair	Combinational with a register ALUT/register pair
Stratix III: EP3SL340F1760C4											
CifiFinal_TopMain_max	27084 (103)	42529 (12)	68071 (0)	209	29	28	142	988	2142 (20)	43163 (0)	24908 (83)

**figura 101** Resultados da melhor compilação e roteamento obtidos.

Após a latência do sistema, a arquitetura proposta é capaz de classificar cada pixel como candidato ou não-candidato a cada ciclo de relógio, indicando também, para cada pixel candidato, qual das 7 escalas de entrada, melhor corresponde ao pixel candidato.

Como cada pixel é processado em apenas um ciclo de *relógio*, o tempo de processamento total é obtido somando-se o número de pixels processáveis na imagem *A* com o número de ciclos de latência para o primeiro preenchimento da matriz CWP em cada linha, ou seja pela equação 4:

$$\frac{(nCol - rCWP) * (nLin - rCWP) + (nLin - 1) * rCWP}{freqClock[MHz]} * 10^{-3} [ms] \quad (4)$$

Onde:

$nCol$  - corresponde ao número de colunas da imagem  $A$ ;

$nLin$  - corresponde ao número de linhas da imagem  $A$ ;

$rCWP$  - corresponde a largura da matriz  $CWP$ ;

$freqRelógio$  - corresponde a frequência de operação da FPGA.

Pela equação (4) e para a frequência de operação obtida para o filtro completo, o tempo de processamento necessário para o filtro Cifi localizar imagens de máscara com uma janela de processamento de 53x53 pixels em uma imagem de 640x480 pixels é de 1,367ms. Este resultado pode ser comparado com o tempo de 7s requeridos pelo algoritmo Ciratefi executado em um computador de 3GHz, ou seja, o algoritmo projetado em *hardware* é cerca de 5100 vezes mais rápido.

## CAPÍTULO 6. DISCUSSÕES

### 6.1. PRECISÃO DOS RESULTADOS

A diferença encontrada entre os valores das médias calculadas em ponto flutuante e aquelas calculadas em ponto fixo não foi significativa. O erro, conforme comparação dos resultados na figura 79 permanece somente nas casas decimais. Trata-se de uma precisão bastante satisfatória, uma vez que a divisão é feita através da multiplicação do inverso (em ponto fixo) do número de pontos de cada círculo, o que incorre em um erro. Este erro pode ser então considerado desprezível.

No item 5.3 discutimos em detalhe o processo de cálculo das médias tomando o intervalo adequado em bits do resultado da multiplicação da soma de pixels por uma constante. Ao se tomar o intervalo de bits do 23 ao 16, seria possível fazer uma análise dos 16 bits menos significativos resultantes desta multiplicação (bit 15 ao bit 0) de modo a implementar algum tipo de arredondamento. No entanto, consideramos que o impacto no resultado final da correlação seria muito pouco relevante, visto que já se tinha alcançado uma precisão bastante razoável para este cálculo das médias.

A tabela 3 faz a correspondência com as figura 87 (coluna 'A' da tabela) e 88 (coluna 'B' da tabela) apresentando respectivamente os resultados do cálculo da correlação em ponto fixo e em ponto flutuante. Nota-se que a diferença entre os resultados obtidos ocorre a partir da segunda casa decimal.

Escala	A	B
0	0,339372	0,315214
1	0,795769	0,772031
2	0,852623	0,844554
3	0,920918	0,914291
4	0,955258	0,952509
5	0,990007	0,986816
6	1,002161	1,000000

**Tabela 3.** Comparação dos resultados das correlações com precisões diferentes.

Este erro se deve à diferença de precisão das variáveis utilizadas nos dois modelos de implementações. Enquanto em *hardware* foi utilizada precisão de 16 bits

para o resultado da correlação, em *software* foi utilizada precisão do tipo *double* que possui 64 bits.

Finalmente, uma análise final do cálculo das correlações foi feita para verificar as diferenças existentes no resultado da divisão (que é o processo final para obtenção da correlação) quando são utilizadas variáveis em ponto-fixo empregando dois métodos para a divisão.

No primeiro método, realizou-se todo o cálculo dos termos da equação de correlação com variáveis do tipo inteiro e uma operação normal de divisão em C. Percebeu-se, no entanto, que este processo gerava resultados com diferenças sutis em relação ao obtido em *hardware*.

Assim, acrescentamos ao programa em C um cálculo de divisão que reproduzisse fielmente o método de cálculo em *hardware*. As diferenças nos resultados obtidos utilizando estes dois métodos de cálculo da divisão podem ser analisadas através da figura 102.

```

-----
IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 0 QUE POSSUI 5 CIRCULOS.
-----
CORRELACAO ESCALA 0 EM FLOAT COM DIVISÃO NORMAL = 0.524887 .
CORRELACAO ESCALA 0 EM FLOAT COM DIVISÃO EM PF = 0.524872 .
.
-----
IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 1 QUE POSSUI 7 CIRCULOS.
-----
CORRELACAO ESCALA 1 EM FLOAT COM DIVISÃO NORMAL = 0.827746 .
CORRELACAO ESCALA 1 EM FLOAT COM DIVISÃO EM PF = 0.827744 .
.
-----
IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 2 QUE POSSUI 8 CIRCULOS.
-----
CORRELACAO ESCALA 2 EM FLOAT COM DIVISÃO NORMAL = 0.878647 .
CORRELACAO ESCALA 2 EM FLOAT COM DIVISÃO EM PF = 0.878647 .
.
-----
IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 3 QUE POSSUI 9 CIRCULOS.
-----
CORRELACAO ESCALA 3 EM FLOAT COM DIVISÃO NORMAL = 0.936863 .
CORRELACAO ESCALA 3 EM FLOAT COM DIVISÃO EM PF = 0.936859 .
.
-----
IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 4 QUE POSSUI 10 CIRCULOS.
-----
CORRELACAO ESCALA 4 EM FLOAT COM DIVISÃO NORMAL = 0.966424 .
CORRELACAO ESCALA 4 EM FLOAT COM DIVISÃO EM PF = 0.966415 .
.
-----
IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 5 QUE POSSUI 12 CIRCULOS.
-----
CORRELACAO ESCALA 5 EM FLOAT COM DIVISÃO NORMAL = 0.994866 .
CORRELACAO ESCALA 5 EM FLOAT COM DIVISÃO EM PF = 0.994858 .
.
-----
IMPRIMINDO RESULTADOS INTERMEDIARIOS EM PONTO FIXO PARA A ESCALA = 6 QUE POSSUI 14 CIRCULOS.
-----
CORRELACAO ESCALA 6 EM FLOAT COM DIVISÃO NORMAL = 0.989727 .
CORRELACAO ESCALA 6 EM FLOAT COM DIVISÃO EM PF = 0.989716 .

```

**figura 102** Resultados do cálculo da divisão da correlação através da divisão em C e reproduzindo o processo de divisão em ponto-fixo executado em *hardware*.

## 6.2. MELHORAMENTOS FUTUROS

Podem ser feitos melhoramentos a fim de se alcançar maiores frequências de operação com o uso de outras técnicas e ferramentas de análise de roteamento e de frequências, disponibilizadas pela própria Altera. Tais ferramentas como o *TimeQuest* e o *Logiclock*, permitem análises mais complexas envolvendo os tempos de propagação dos circuitos da FPGA implementados na etapa de roteamento, além de otimizações no âmbito do leiaute modular interno da FPGA. Utilizadas adequadamente, estas ferramentas devem trazer ganhos de desempenho significativos, aumentando a frequência de operação dos módulos individualmente e do sistema completo.

Além de otimizações envolvendo o roteamento no dispositivo StratixIII, algumas outras melhorias e considerações são discutidas nos tópicos seguintes.

### 6.2.1. TERMOS CONSTANTES NA CORRELAÇÃO

As médias dos círculos de  $Q$  são pré-calculados em *software* e transferidos ao próprio código VHDL. Porém como tais parâmetros são entrada para os módulos de cálculo, ao alterar-se a imagem de máscara  $Q$ , basta transferir os novos valores ao mesmo *hardware* já compilado, através dos próprios pinos de E/S da FPGA ou de um sistema em mais alto nível, como uma CPU tipo Nios II, instanciada dentro da própria FPGA.

Portanto, o *hardware*, uma vez gerado, compilado, roteado e executado em uma FPGA, não precisará mais ser alterado, mesmo para diferentes imagens  $Q$ , salvo mudanças nos parâmetros de geração (resolução da matriz CWP e quantidades de círculos ou escalas).

Nos códigos implementados neste trabalho, somente o valor de  $y_i$ , que corresponde a média dos pixels de determinado círculo da imagem  $Q$ , foi pré-calculado em *software* e fornecido ao *hardware* na própria geração do código VHDL correspondente.

Na figura 103, os valores em círculos indicam os cálculos que também podem ser pré-calculados em *software*. Os valores com um “traço” inclinado indicam os cálculos intermediários que podem passar a ficar a cargo do *software gerador de códigos VHDL*. Os cálculos indicados em retângulos são valores que já estarão prontos para o *hardware* no momento do cálculo em paralelo.

Esta nova abordagem deve proporcionar ao sistema completo uma economia significativa no consumo de lógica do processador FPGA. Por exemplo, economiza-se um módulo completo de raiz quadrada de *hardware* (indicado pelo retângulo na etapa 5 da mesma figura).

Os valores calculados em *software* se tornarão assim, entrada para os módulos em *hardware* e deverão ser adequadamente ligados e instanciados.

The diagram illustrates the steps of correlation calculations, with annotations indicating which parts can be pre-calculated in software (circles), which are intermediate steps for the software generator (slashes), and which are final hardware inputs (rectangles).

$$\begin{array}{l}
 \sum x_i \\
 \sum x_i^2 \\
 \sum y_i \\
 \sum y_i^2 \\
 \sum x_i y_i
 \end{array}
 \begin{array}{l}
 \sum x_i \sum y_i \\
 n \sum x_i y_i \\
 (\sum y_i)^2 \\
 n \sum y_i^2 \\
 (\sum x_i)^2
 \end{array}
 \begin{array}{l}
 n \sum x_i y_i - \sum x_i \sum y_i \\
 n \sum x_i^2 - (\sum x_i)^2 \\
 n \sum y_i^2 - (\sum y_i)^2
 \end{array}
 \begin{array}{l}
 \sqrt{n \sum x_i^2 - (\sum x_i)^2} \\
 \sqrt{n \sum y_i^2 - (\sum y_i)^2} \\
 \sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}
 \end{array}
 \begin{array}{l}
 \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}
 \end{array}$$

(1) (2) (3) (4) (5) (6)

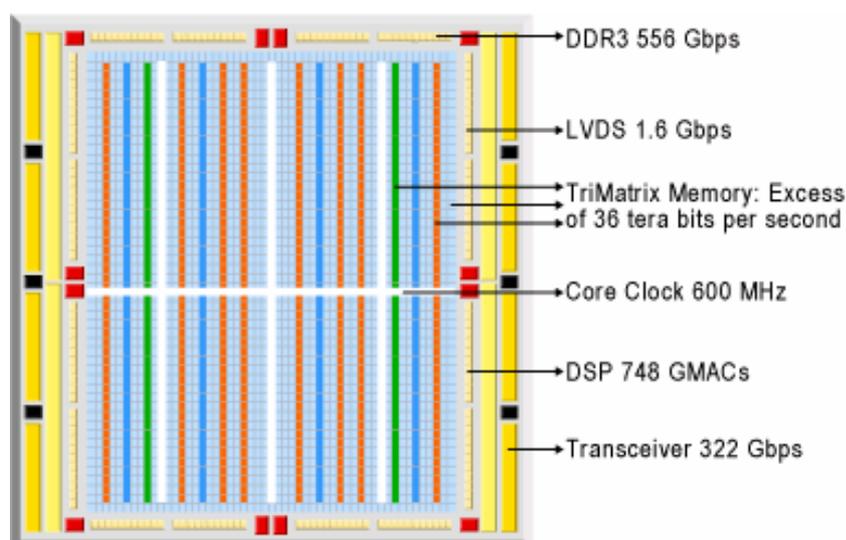
**figura 103** Etapas dos cálculos das correlações indicando os valores constantes que podem ser calculados em *software* enviando-os como entrada ao *hardware*.

## 6.2.2. FPGAs DE ALTO DESEMPENHO

Neste tópico abordamos os melhoramentos que podem ser feitos utilizando novos dispositivos FPGA que estão sendo disponibilizados atualmente no mercado. Primeiramente citamos o lançamento das novas versões de dispositivos da família de alto desempenho da Altera, a Stratix IV [73] e suas principais diferenças em relação à família de dispositivos utilizadas neste trabalho, a Stratix III. No tópico seguinte, realçamos a importância do surgimento de FPGAs com sistemas de *pipeline* integrados ao silício do dispositivo, em início de comercialização e que podem vir a representar um marco na tecnologia.

### 6.2.2.1. STRATIX IV

A Altera iniciou a comercialização dos dispositivos FPGA da família Stratix IV no segundo bimestre de 2008 marcando a entrada dos dispositivos FPGA nas dimensões de 40nm. Segundo a empresa, dispositivos com tal escala dimensional contribuem para o aumento da densidade lógica dos dispositivos, transistores de melhor desempenho e uma diminuição do consumo [72].



**figura 104** Diagrama de representação dos dispositivos da família Stratix IV da Altera [33].

Segundo o fabricante, na tecnologia de 40nm, as FPGA Stratix IV possuem transistores com o comprimento da porta sendo 38,5% menor do que aqueles da tecnologia de 65nm e 11% menor do que aqueles da tecnologia de 45nm. E além disso houve uma melhora de até 30% na mobilidade de elétrons com a tecnologia.

De forma resumida, as principais características dos dispositivos Stratix IV[73] são:

- Alta velocidade dos *tranceivers*. De até 8,5 Gbps;
- Interface com memórias DDR3 em 1.067Mbps (533Mhz);
- Desempenho dos módulos DSP de até 550MHz através da otimização; do paralelismo na relação DSP/memória/lógica;
- Canais LVDS de 1.6Gbps;
- Velocidade dos relógios de 600MHz.

As FPGAs Stratix IV utilizam os módulos *Adaptive Logic Modulo* (ALM) para implementar funções lógicas e sua eficiência se deve ao particionamento destes ALMs [74].

### 6.2.2.2. FPGAs ACHRONIX

A empresa *Achronix Semiconductor* é um fabricante de FPGAs que surgiu recentemente no mercado e que se diferencia dos demais, por oferecer FPGAs com desempenho muito superior aos atuais, chegando a frequências de 1.5GHz.

Como comentado no item 3.5.1, ressaltamos primeiramente a importância de técnicas de programação para processamentos em *pipeline* e também como este conceito começa a ser integrado na eletrônica física dos dispositivos FPGA fabricados pela Achronix. A empresa utiliza técnicas assíncronas para aumentar a velocidade da propagação dos sinais lógicos e é originária dos laboratórios da *Cornell University*.

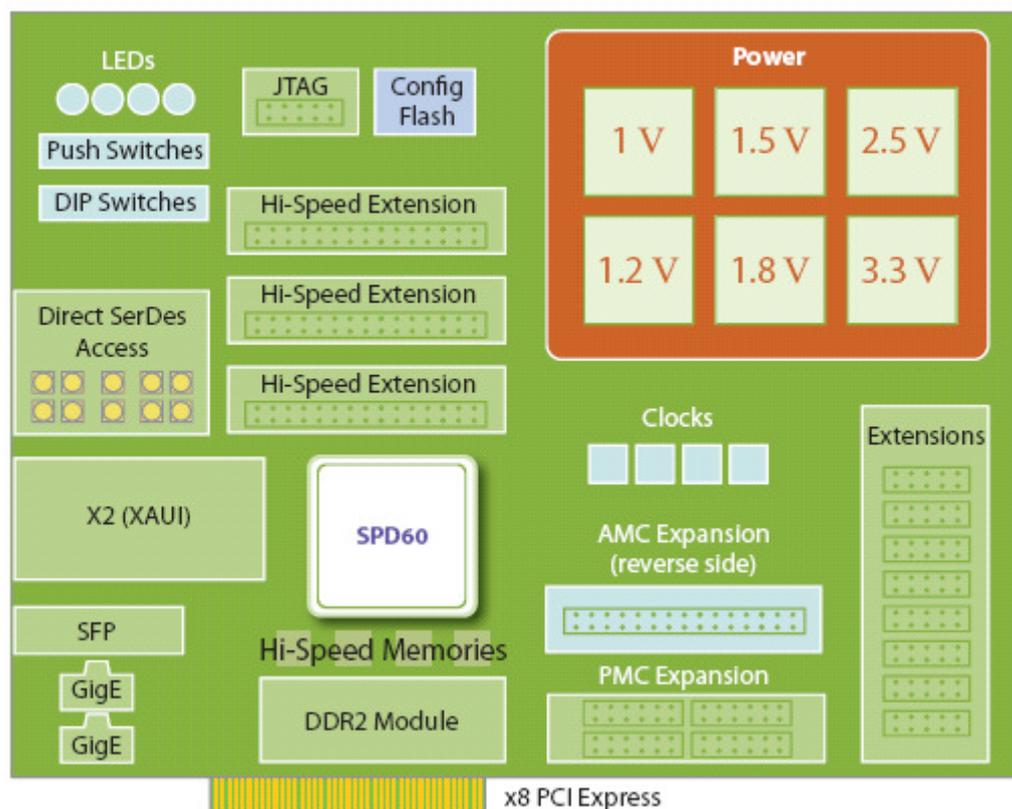
Os primeiros dispositivos da Achronix foram chamados de *Speedster* e se baseiam em matrizes lógicas tipo *look-up table*. Ferramentas de síntese e simulação comuns ao mercado (Synmplicity/Synopsys ou Mentor Graphics, por exemplo) já podem operar com os dispositivos *Speedster*.

Os dispositivos Achronix se assemelham aos FPGAs tradicionais exceto pela diferença chave, que é a interconexão em *pipeline* entre as LUTs. Essencialmente cada ponto dentro do barramento interno possui uma FIFO de 1 bit. Este tipo de abordagem simplifica enormemente a distribuição de *relógios*, que é o maior problema em FPGAs com elevada quantidade de lógica.

A empresa também já disponibiliza um kit de desenvolvimento, o SPD60 [75] cujo diagrama funcional está ilustrado na figura 105. Este kit de desenvolvimento se destaca pela sua elevada capacidade de processamento, com diversos periféricos para este fim como memórias e conectores para trocas de dados em alta velocidade. Em destaque, o barramento *PCI Express*.

A Achronix afirma que é possível operar a taxas de até 1,5GHz. A título de comparação, se o Filtro Cifi em *hardware* projetado neste trabalho pudesse ser executado a 1GHz (66% da frequência máxima de operação afirmada pelo fabricante) **o filtro processaria uma imagem completa de 640x480 em 0,276 ms ou 23.359 vezes mais rápido do que em computadores convencionais.**

## SPD60 Development Board



**figura 105** Diagrama de blocos funcional dos periféricos presentes no kit de desenvolvimento da empresa Achronix [77].

### 6.2.3. IDEALIZAÇÃO DO SISTEMA FINAL

Em [70] e [76] foi implementado um sistema autômato com capacidade de identificação de imagens utilizando FPGAs como co-processadores. Em [51] os autores implementaram um sistema (também com processamento híbrido) de busca e localização de padrões em imagens 3D para aplicações médicas utilizando Transformada de Fourier Rápida.

Neste trabalho nos preocupamos com o aspecto teórico e quantitativo para a viabilidade de sistemas de processamento de imagens de alto desempenho. Uma abordagem importante é analisar algumas opções de infra-estrutura física capazes de operacionalizar os sistemas propostos nesta dissertação.

O objetivo permanece o de superar o período de quadros por segundo viabilizando detecção de imagens de máscaras em tempo real. Para isto é preciso que o tempo de detecção não ultrapasse os 33ms. Como o tempo de detecção simulado neste projeto, para uma imagem de 640x480 está ainda cerca de 20 vezes mais rápido do que o necessário, a diferença de tempo pode ser usada para toda a troca de informações entre os dispositivos presentes em tal sistema, mais especificamente, na troca e acesso aos dados nas memórias externas e internas à FPGA.

Em um sistema executando o algoritmo Ciratefi em tempo real, todo o *hardware* é gerado pelo programa em C. Estes códigos serão sintetizados e gravados apropriadamente na FPGA. Este processo não é trivial e pode ser considerado um projeto a parte. Entretanto, uma vez roteado o *hardware* para a FPGA não será mais possível alterar os parâmetros mencionados a seguir, sem ter que recopilar todo o projeto para os parâmetros novos. A re-compilação de um sistema deste nível de complexidade pode levar algumas horas, mesmo em PCs de multi-processadores.

Estas parametrizações referem-se à resolução da imagem  $A$ , da janela de processamento CWP utilizada, o número de círculos tratados na CWP, o número de escalas das máscaras e os fatores de escala. Porém, o sistema terá bastante praticidade para poder processar e analisar imagens  $A$ , em tempo real e ser alterado para diferentes imagens de máscara; tudo isso sem precisar re-sintetizar ou re-rotear o *hardware*.

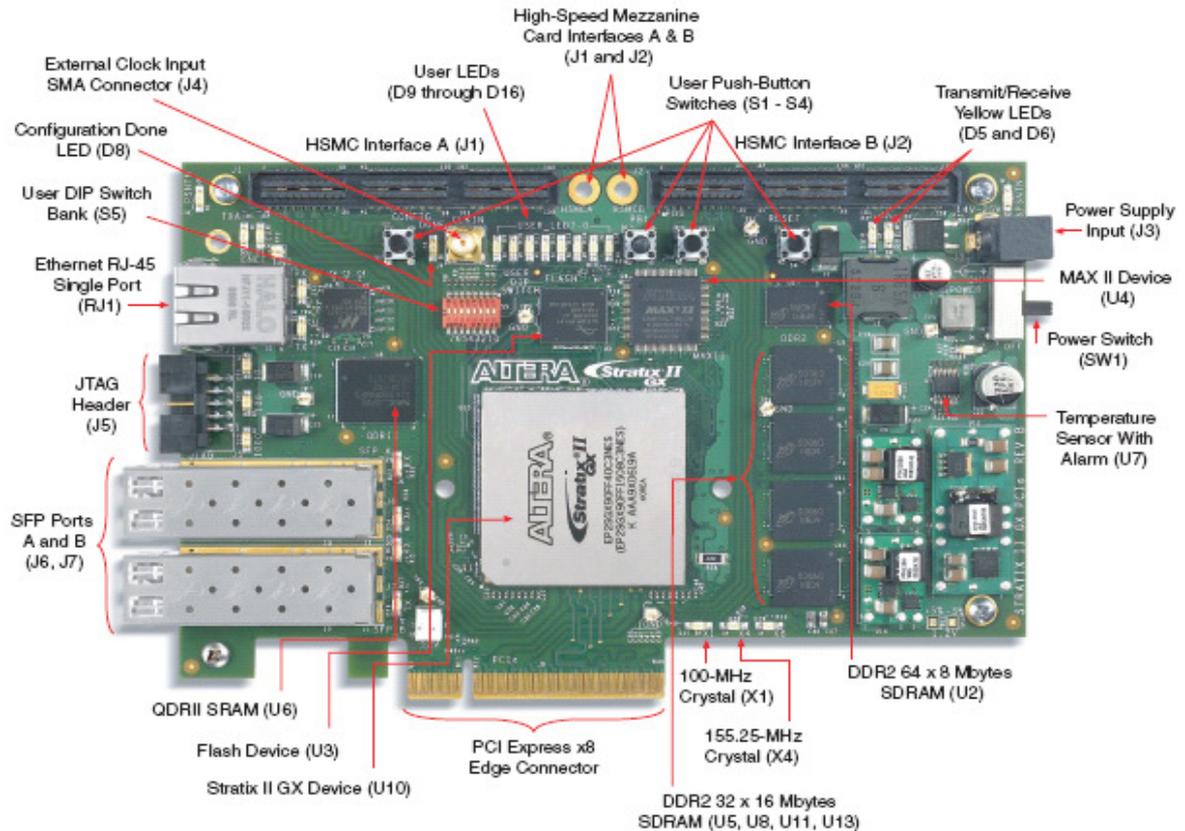
No sistema ideal o *hardware* seria o responsável por resolver os filtros Cifi e Rafi. Via um barramento de comunicação de alto desempenho com o PC, onde o filtro Tefi seria executado, o último filtro receberia as informações de quais são os pixels candidatos do segundo grau e faria o casamento final de imagens.

### **6.2.3.1. BARRAMENTOS *PCI EXPRESS***

*PCI Express* é um padrão de interconexão de E/S de propósito geral e alto desempenho utilizado em diversas plataformas de computação e comunicação. A ligação *PCI Express* fundamental, consiste de dois pares diferenciais tipo “baixa-tensão”: um par de transmissão e outro de recepção [77].

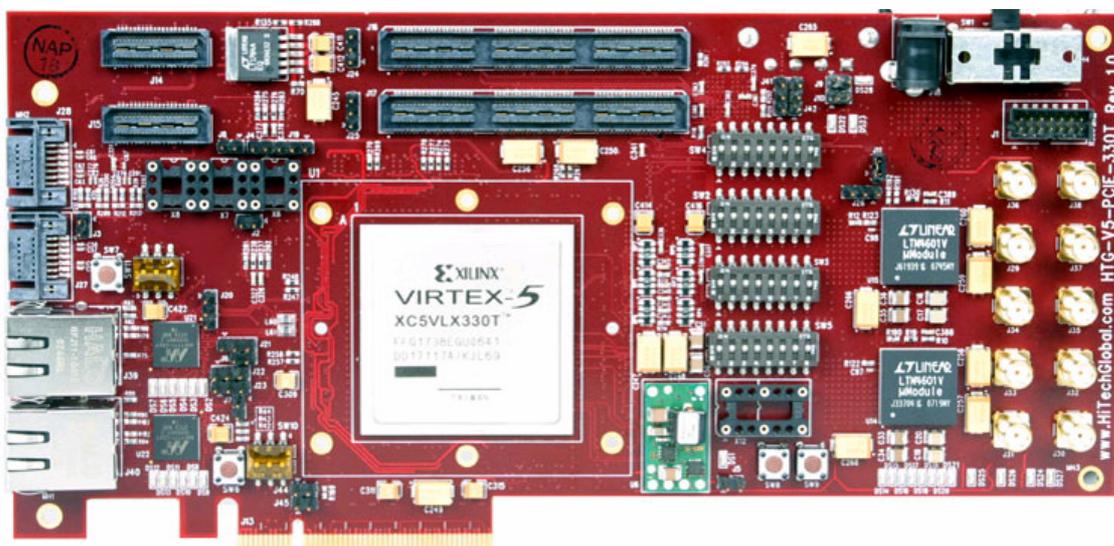
Um barramento de alto desempenho tipo *PCI* não garante, necessariamente, uma alta taxa de transferência de dados entre o co-processador FPGA e a CPU. A razão do desempenho obtido na prática por determinada aplicação depende fortemente dos *drivers* dos dispositivos [14].

No momento do desenvolvimento deste trabalho, a Altera somente dispunha de um kit de desenvolvimento com *PCI Express* para os dispositivos da família Stratix II (figura 106). Os dispositivos da família Stratix IV possuem circuitos de alto desempenho dedicados, em silício, para operação com interfaces *PCI Express* [78] mas até o momento não há um kit de desenvolvimento *PCI Express* com FPGAs da Stratix IV disponível.



**figura 106** Foto da vista superior para o kit de desenvolvimento *PCI Express Stratix II GX Development Board* [79].

A empresa Xilinx possui também um kit de desenvolvimento com *PCI Express* para sua família de FPGAs de alto desempenho, a *Virtex-5* [80], fabricada pela empresa HitechGlobal [81]. Maiores detalhes técnicos sobre o kit de desenvolvimento podem ser encontrados em [82].



**figura 107** Kit de Desenvolvimento *PCI Express* com FPGA da Xilinx [84].

## 6.2.4. FILTRO CIFI EM TEMPO REAL COM CÂMERA

O objetivo desta melhoria é testar na prática e em tempo real o *hardware* desenvolvido nesta dissertação, utilizando o kit de desenvolvimento Stratix III [83]. O sistema de testes proposto possui o objetivo similar ao implementado pelos autores em [17], no qual foi desenvolvido um sistema de entrada de pixels através de uma câmera com taxa de 30 quadros por segundo e imagens com resolução de 320x240.

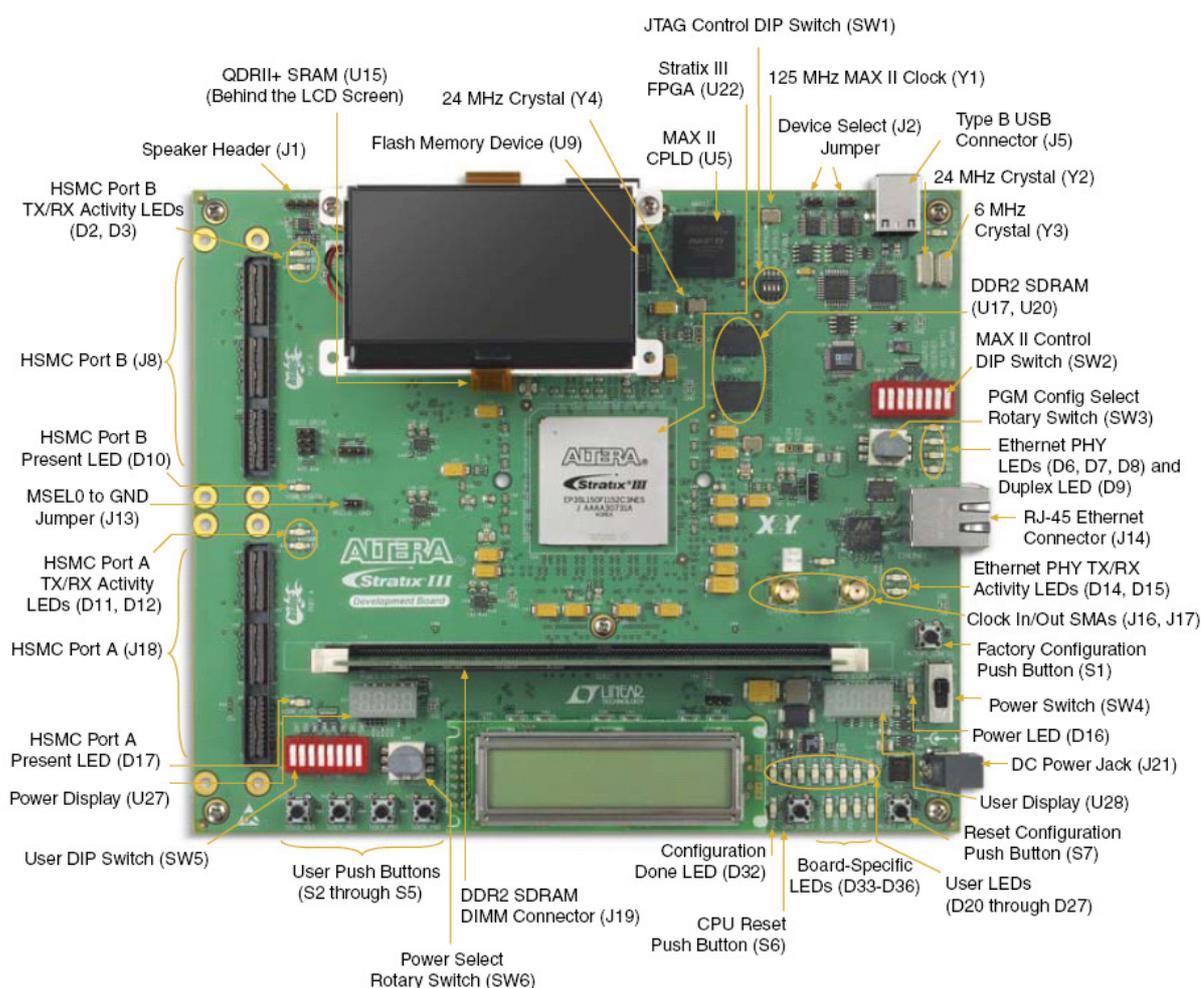
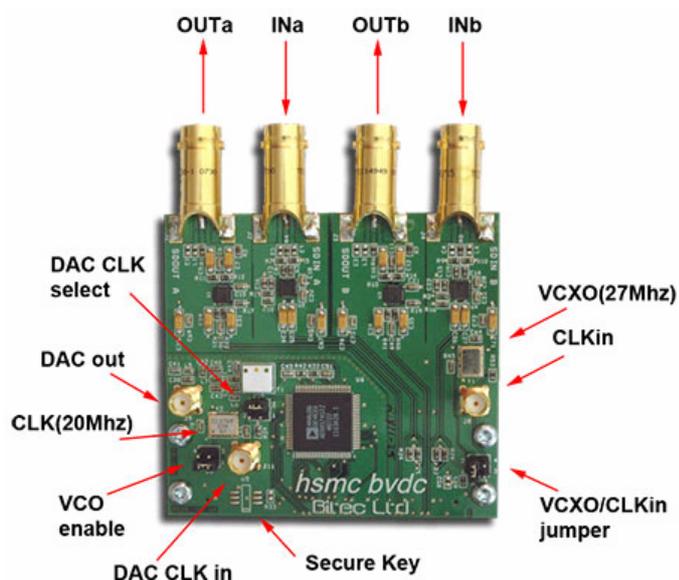


figura 108 Kit de desenvolvimento Stratix III [85].

O número de leds, presente no kit, a serem acesos, podem corresponder à quantidade de pixels candidatos encontrados no quadro. O LCD alfa numérico é capaz de apresentar os resultados do algoritmo Cifi indicando as coordenadas e a escala dos pixels candidatos. Um dos conectores HSMC pode ser usado para conexões de kits tipo “placas-filha” com câmeras de vídeo. Na figura 109 tem-se um

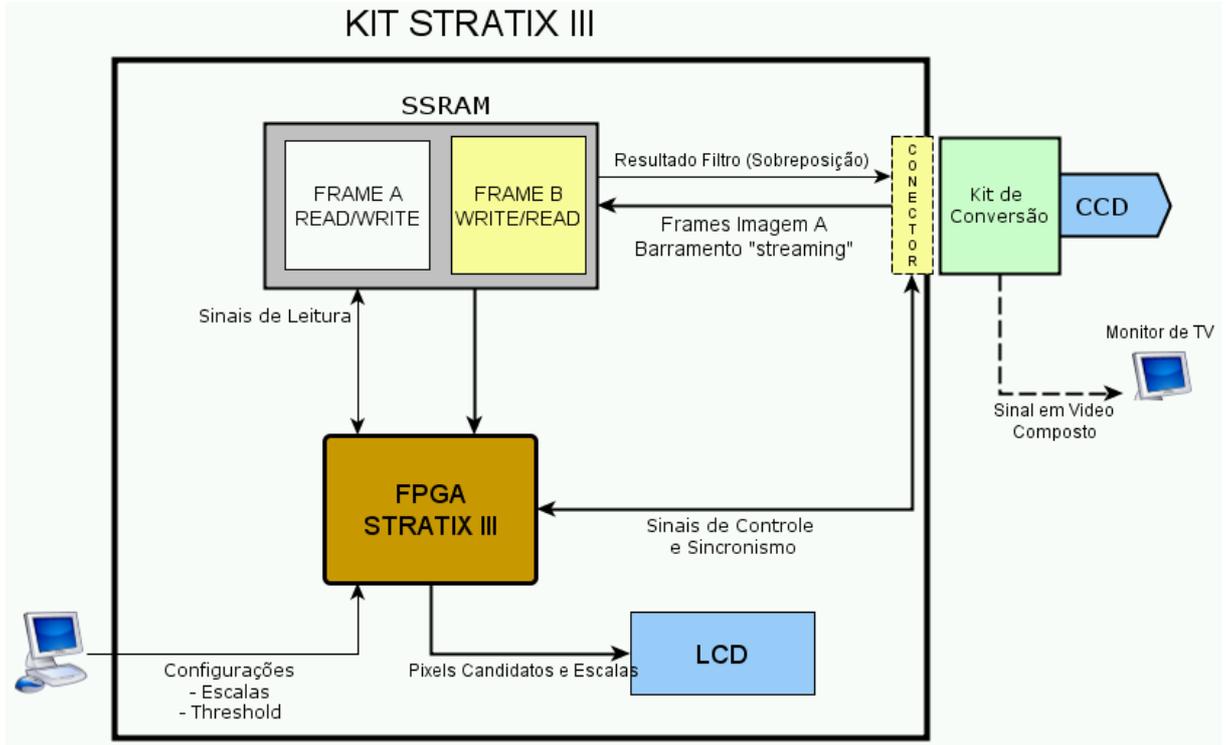
exemplo deste tipo de placa, fabricado pela empresa Bitec [84], com entrada para vídeo composto e conexão com o kit Stratix III sugerido.



**figura 109** Placa com entrada de vídeo composto e conector tipo HSMC para conexão com os kits de desenvolvimento da Altera [86].

Como a placa de entrada de vídeo na figura 109 possui entrada e saída de vídeo, é desejável também que o resultado do processamento do filtro seja disponibilizado em outro monitor, onde os pixels candidatos (eventualmente indicados por pixels coloridos) seriam sobrepostos à imagem recebida pela câmera.

A figura 110 apresenta um diagrama de blocos da arquitetura do sistema sugerido com a entrada dos quadros por uma câmera convencional monocromática de resolução 640x480. O processamento é feito pela FPGA. A comunicação com o PC é feita para a configuração dos parâmetros e obtenção dos resultados em arquivos.



**figura 110** Diagrama de blocos para a estrutura de execução do filtro Cifi em tempo real.

## 6.2.5. CONSIDERAÇÕES PARA IMPLEMENTAÇÃO DO FILTRO RAFI

Embora apresentada uma proposta de solução para o segundo filtro do algoritmo Ciratefi, o filtro Rafi não foi implementado neste trabalho de dissertação. De modo semelhante aos programas em C feitos para o filtro Cifi, deve-se resumidamente realizar as seguintes tarefas para a implementação do filtro Rafi:

- 1) Pré-processamento das sete imagens de máscara. Devem ser calculados os sete vetores  $R_Q$  correspondentes a cada escala. Cada um destes vetores possui comprimento de 36 elementos que correspondem às 36 linhas radiais. Cada um dos elementos do vetor possui a média dos valores dos pixels contidos nos raios sobrescritos a cada escala diferente, para cada um dos 36 raios traçados em todas as diferentes escalas da imagem de máscara.
- 2) Geração das coordenadas dos pontos que irão compor as retas. Estas coordenadas serão atribuídas à matriz de processamento que irá realizar a somatória dos pontos (janela de processamento CWP).
- 3) Deve-se contar o número de pixels presentes em cada linha radial. Este valor pode exceder em algumas unidades, o raio correspondente para cada escala e o *software* gerador de VHDL deve gerenciar o problema uniformizando o diâmetro dos raios. Obtida esta quantidade, deve-se calcular o seu inverso em ponto-fixado. Esta constante será utilizada no módulo CWP para o filtro Rafi, para realizar o cálculo das médias dos valores dos raios. É importante notar que cada uma das 36 linhas radiais pode assumir qualquer uma das 7 escalas. Portanto, cada linha radial possuirá, na verdade, 7 constantes diferentes. O sistema em *hardware* deve ser projetado de modo a multiplexar qual das constantes e coordenadas de somatórias serão utilizadas conforme o correspondente valor de escala do pixel candidato, recebido do filtro Cifi.

## CONCLUSÃO

Neste trabalho, projetamos um sistema em FPGA que implementa parte de um algoritmo de localização de imagens de máscara. Este algoritmo, chamado de Ciratefi, realiza a busca por padrão de imagens de forma invariante em relação à escala, rotação, translação, brilho e contraste. O algoritmo é dividido em três filtros que sucessivamente selecionam os filtros que têm maior chance de pertencerem à imagem buscada.

O primeiro destes filtros é o chamado filtro Cifi que, independentemente da rotação e da translação, seleciona os pixels com chance de pertencerem à imagem ambiente selecionada. O segundo filtro, chamado de filtro Rafi, filtra os pixels selecionados pela etapa anterior e detecta o melhor ângulo de rotação na imagem. A análise final, a qual é independente de brilho e contraste, é feita pelo último filtro, chamado de filtro Tefi, muito mais veloz do que os demais filtros e por isso não foi intenção deste trabalho projetar uma solução para este último filtro em *hardware*.

Como mencionado no item 6.2.3 o sistema final pode ser composto por um sistema eletrônico híbrido com processamento paralelo em *hardware* para os dois primeiros filtros e processamento convencional para o filtro final.

Este trabalho teve, portanto, como objetivo, propor a solução em *hardware* de alto desempenho, para o primeiro filtro do algoritmo Ciratefi. Este *hardware* desenvolvido deveria ser capaz de obter resultados com precisão satisfatória e com o desempenho maior possível, viabilizando inclusive, aplicações em tempo real.

Foram implementados, testados e validados o programa em C gerador de códigos VHDL e os módulos em *hardware* apenas para o filtro Cifi. Embora o projeto do *hardware* para o segundo filtro, o filtro Rafi, tenha sido concluído e que suas soluções matemáticas em *hardware* sejam muito similares às do trabalho aqui desenvolvido, a implementação do programa em C gerador de *hardware*, seus módulos de teste, as validações e ajustes de sincronismo, representam uma quantidade significativa de trabalho ainda a ser realizado.

Para se atingir ambos requisitos de flexibilidade e alto desempenho \_\_ requisitos estes, comumente antagônicos \_\_ criamos programas em linguagem C que geram automaticamente módulos em VHDL para os parâmetros selecionados. O sistema resultante proposto tem seu desempenho maximizado a medida em que é

capaz de classificar um pixel por ciclo de *relógio* requerendo 1.36ms para processar um quadro completo de 640x480, aproximadamente 5100 vezes mais rápido do que a implementação em *software*. Como sugerido ao longo do item 6.2, desempenhos do *hardware* roteado ou de dispositivos mais rápidos, que atinjam frequências de operação mais elevadas, irão proporcionalmente baixar o tempo de processamento para um quadro.

## ARTIGOS GERADOS NO MESTRADO

H. P. A. Nobre, H. Y. Kim. "Automatic VHDL Generation For Solving Rotation and Scale-Invariant Template Matching in FPGA". *SPL 2009 – V Southern Conference on Programmable Logic*, São Carlos, Brazil, April/2009. (artigo aceito e eleito melhor artigo do congresso)

H. P. A. Nobre, H. Y. Kim. "Real-Time FPGA Implementation of Rotation And Scale-Invariant Template Matching,". *SPIE 2009 – SPIE Optical Engineering and Applications*, San Diego, USA, August/2009. (artigo aceito)

H. P. A. Nobre, H. Y. Kim. "Automatic VHDL Generation For Solving in Real-Time a Rotation Invariant Template Matching Filter in FPGA,". Artigo em preparação com a implementação do filtro Rafi a ser submetido em revista especializada em processamento em tempo-real.

## REFERÊNCIAS BIBLIOGRÁFICAS

- 
- <sup>1</sup> H. Y. Kim and S. A. Araújo. "Grayscale Template-Matching Invariant to Rotation, Scale, Translation, Brightness and Contrast". *IEEE Pacific-Rim Symposium on Image and Video Technology, Lecture Notes in Computer Science*, vol. 4872, pp. 100-113, 2007.
- <sup>2</sup> S. A. Araújo, H. Y. Kim. "Rotation, scale and translation-invariant segmentation-free grayscale shape recognition using mathematical morphology". *Int. Symposium on Mathematical Morphology*, 2007.
- <sup>3</sup> Z. Bin Ibrahim. *Printed Circuit Board Inspection Using Wavelet-Based Technique*. Universiti Teknologi Malaysia. Tese de mestrado defendida em 2002.
- <sup>4</sup> S. Se, T. Barfoot, P. Jasiobedzki. "Visual Motion Estimation And Terrain Modeling For Planetary Rovers". SAIRAS - *The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*. European Space Agency SP-603, pp.101.1, 2005.
- <sup>5</sup> C. Arth, C. Leistner, H. Bischof. "Using Robust Local Features on DSP-Based Embedded Systems". *Embedded Computer Vision, Advances in Pattern Recognition*, vol. ISBN 978-1-84800-303-3, pp. 79, 2009.
- <sup>6</sup> M. Uenohara, T. Kanade. "Decomposition for Fast Pattern Matching With a Large Set of Templates". *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 19, No. 8, 1997.
- <sup>7</sup> Z. K. Baker, M. B. Gokhale, J. L. Tripp. "Matched Filter Computation on FPGA, Cell and GPU". *International Symposium on Field-Programmable Custom Computing Machines. 15th Annual IEEE Symposium on Volume*. Issue, 23-25, pp. 207 – 218, 2007.
- <sup>8</sup> L. Ding, A. Goshtasby, M. Satter. "Volume image registration by template matching". *Image and Vision Computing*. Issue 12, vol. 19, pp. 821-832, 2001.
- <sup>9</sup> D. Casasent, D. Psaltis. "New Optical Transforms for Pattern and Recognition". *Proceedings of IEEE*. vol. 65, No. 1, pp.77–84, 1977.
- <sup>10</sup> Q. Chen, M. Defries, F. Deconinck. "Symmetric Phase-Only Matched Filtering of Fourier". *Mellin Transforms for Image Registration and Recognition, IEEE Trans*, vol.16, No.12, pp.1156–1168, 1994.
- <sup>11</sup> H. Onishi, H. Suzuki. "Detection of Rotation and Parallel Translation Using Hough and Fourier Transforms," *Proc. 1996 Int. Conf. on Image Processing*, vol.3, pp.827–830, 1996.
- <sup>12</sup> H. D. Ballard. "Generalizing the Hough Transform to Detect Arbitrary Shapes". *Pattern Recognition*, vol. 13, no. 2, pp.111–122, 1981.
- <sup>13</sup> N. Bauer, R. W. Frischholz. "3-D automatic motion analysis". *SATA 28th Symposium, Proceedings on Robotics, Motion and Machine Vision in the Automotive Industries, Stuttgart*. Craydon, England: Automotive Automation Limited, 1995.
- <sup>14</sup> B. Jahne, H. HauBecker, P. GeiBler. *Handbook of Computer Vision and Applications*. Academic Press Systems and Applications, vol. 3, 1999.
- <sup>15</sup> S. Hezel, A. Kugel, R. Manner, D. M. Gavrilu. "FPGA-based Template Matching using Distance Transforms". *Proceedings of the 10 th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 1082-3409, 2002.
- <sup>16</sup> M. Shen, H. Song, W. Sheng, Z. Liu. "Fast Correlation Tracking Method based on Circular Projection". Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel / Distributed Computing. pp. 235-238, 2007.

- 
- <sup>17</sup> V. Bonato, E. Marques, G. A. Constantinides. "A Parallel Hardware Architecture for Image Feature Detection". *Lecture Notes in Computer Science*, pp 137-148, 2008.
- <sup>18</sup> C. Harris, M. J. Stephens. "A combined corner and edge detector". *Proceedings of the Avley Vision Conference*, pp. 147-152, 1988.
- <sup>19</sup> D. Lowe. "Distinctive image features from scale-invariant keypoints". *International Journal of Computer Vision*, v. 60, n. 2, p.91-110, 2004.
- <sup>20</sup> V. Andersen, L. Pellarin, R. Anderson. "Scale-Invariant Feature Transform (SIFT): Performance and Application," 2006. Documento acessado por <http://www.itu.dk/~vedrana/reports/SIFT.pdf>. Acesso em janeiro de 2009.
- <sup>21</sup> N. M. Duarte. "Coprocesador criptográfico Advanced Encryption Standard (AES) baseado em lógica programável". Tese de Mestrado, USP, São Paulo, 2004.
- <sup>22</sup> S. Brown, Z. Vranesic. *Fundamentals of digital logic with VHDL design*. 1.ed. EUA: McGraw Hill, 2000.
- <sup>23</sup> Especificações técnicas dos dispositivos FPGA da família Stratix III <http://www.altera.com/products/devices/stratix-fpgas/stratix-iii/st3-index.jsp>. Acesso em dezembro de 2008.
- <sup>24</sup> Documento da Altera anunciando a tecnologia de 40nm para os dispositivos da família de alto desempenho Stratix IV <http://www.altera.com/literature/br/br-stratix-iv-hardcopy-iv.pdf>. Acesso em dezembro de 2008.
- <sup>25</sup> R. Zeidman. "Introduction to CPLD and FPGA Design". Artigo disponibilizado pela revista técnica *TechOnline*. <http://www.techonline.com/learning/techpaper/193103863>. Acesso em janeiro de 2009.
- <sup>26</sup> M. J. Smith. *Application-Specific Integrated Circuits*. 1.ed. New York: Addison-Wesley Professional, 1997.
- <sup>27</sup> Documento da Altera com especificações do dispositivo CPLD da família MAX. <http://www.altera.com/literature/ds/m3000a.pdf>. Acesso em dezembro de 2008.
- <sup>28</sup> Sítio da empresa Altera, um dos maiores desenvolvedores e fabricantes de FPGA e softwares de projeto para seus dispositivos. [www.altera.com](http://www.altera.com). Acesso em março de 2008.
- <sup>29</sup> Histórico do fabricante de dispositivos programáveis, Xilinx. <http://www.xilinx.com/company/history.htm>. Acesso em janeiro de 2009.
- <sup>30</sup> P. K. Chan, S. Mourad. *Digital design using Field-Programmable Gate Arrays*. 1.ed. New Jersey: Prentice Hall Inc., 1994.
- <sup>31</sup> Apresentação técnica sobre as questões básicas do funcionamento de FPGAs. <http://cas.web.cern.ch/cas/Sweden-2007/Lectures/Web-versions/Serrano-1.pdf>. Acesso em janeiro de 2009.
- <sup>32</sup> Documento técnico e especificações das FPGAs da família Stratix III da Altera. [www.altera.com/literature/hb/stx3/stratix3\\_handbook.pdf](http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf). Acesso em outubro de 2008.
- <sup>33</sup> J. Ferruz, A. Ollero. "Integrated real-time vision system for vehicle control in non-structured environments". *Engineering Applications of Artificial Intelligence*, pp. 215-236, 2000.
- <sup>34</sup> M. Kolsch, S. Butner. "Hardware Considerations for Embedded Vision Systems". *Advances in Pattern Recognition 78-100*, pp. 2-26, 2008.

- 
- <sup>35</sup> C. Torres-Huitzil, M. Arias-Estrada. "Real-time image processing with a compact FPGA-based systolic architecture". *Real-Time Imaging*, pp.177-187, 2004.
- <sup>36</sup> T. Kean, A. Duncan. "A 800 Mpixel/sec Reconfigurable Image Correlator on XC6216". *Proceedings of FPL'97*, pp. 382-391, 1997.
- <sup>37</sup> T. VanCourt, M.C. Herbordt, R.J. Barton. "Case Study of a Functional Genomics Application for an FPGA-Based Coprocessor". *Proceedings of Field Programmable Logic and Applications (FPL)*, pp. 365-374, 2003.
- <sup>38</sup> P. D. Michailidis, K. G. Margaritis. "A programmable array processor architecture for flexible approximate string matching algorithms". *J. Parallel Distrib. Comput.*, pp. 131 - 141, 2007.
- <sup>39</sup> T. VanCourt. Universidade de Boston. Engenharia Elétrica e da Computação. Documento explicando a linha de pesquisa para o desenvolvimento de ferramentas em alto nível para o projeto de sistemas de *hardware*. <http://people.bu.edu/tvancour/research.pdf>. Acesso em janeiro de 2009.
- <sup>40</sup> R. Rakvic, R. Broussard, D. Etter, L. Kennell, J. Matey. "Iris matching for rapid identification with configurable hardware". *SPIE Newsroom*, 2008.
- <sup>41</sup> Manual do usuário do kit de desenvolvimento DE2 da Altera [ftp://ftp.altera.com/up/pub/Webdocs/DE2\\_UserManual.pdf](ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf). Acesso em janeiro de 2009.
- <sup>42</sup> H. Meng, N. Pears, M. Freeman, C. Bailey. "Motion History Histograms for Human Action Recognition". *Advances in Pattern Recognition*, pp. 139-162, 2008.
- <sup>43</sup> Informações técnicas sobre as FPGAs da família Spartan-3 da Xilinx <http://www.xilinx.com/products/spartan3a/>. Acesso em janeiro de 2009.
- <sup>44</sup> K. Ambrosch, M. Humenberger, W. Kubinger, A. Steininger. "SAD-Based Stereo Matching Using FPGAs," *Advances in Pattern Recognition*, pp. 121-138, 2008.
- <sup>45</sup> T. J. Desai, K. J. Hintz. "Volumetric signal processing hardware acceleration for mine detection". *Proc. SPIE The International Society for Optical Engineering*, vol. 5089, pp. 863, 2003.
- <sup>46</sup> Empresa sul-coreana fabricante de sistemas de visão em FPGA, <http://visionst.visualnetworks.kr> Acesso em janeiro de 2009.
- <sup>47</sup> W. J. MacLean. "An Evaluation of the Suitability of FPGAs for Embedded Vision Systems". *Computer Vision and Pattern Recognition - Workshops. IEEE Computer Society Conference*, vol. 25-25, pp. 131, 2005.
- <sup>48</sup> S. Yamamoto, S. Hirai. "Robust and Video-frame Rate Tracking of Planar Motion by Matched Filtering on FPGA". *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, pp. 4707- 4712, 2004.
- <sup>49</sup> Artigo técnico-comercial da Altera "Using PLDs for High-Performance DSP Applications," [http://www.altera.ru/Disks/Altera%20Documentation%20Library/literature/wp/wp\\_dsp\\_app.pdf](http://www.altera.ru/Disks/Altera%20Documentation%20Library/literature/wp/wp_dsp_app.pdf). Acesso em agosto de 2008.
- <sup>50</sup> Ekas. P. "FPGAs rapidly replacing high-performance DSP capability". <http://www.dsp-fpga.com/articles/ekas/>. Acesso em janeiro de 2009.
- <sup>51</sup> T. VanCourt, M. Herbordt. "Practical 3D template matching with FPGAs". *SPIE Newsroom – SPIE. The International Society for Optical Engineering*, 2006. Artigo acessado pelo sítio [http://spie.org/documents/Newsroom/Imported/0013/13\\_248\\_0\\_2006-02-26.pdf](http://spie.org/documents/Newsroom/Imported/0013/13_248_0_2006-02-26.pdf) Acesso em janeiro de 2009.

- 
- <sup>52</sup> Kilts, Steve. *Advanced FPGA Design Architecture, Implementation, and Optimization*. EUA. IEEE. John Wiley & Sons, Inc., Publication, 2007.
- <sup>53</sup> Documento com detalhes da tecnologia de fabricação para FPGAs de maior desempenhos do mercado da empresa Achronix.  
[http://www.achronix.com/index.php?option=com\\_content&task=view&id=34&parentid=47&Itemid=48](http://www.achronix.com/index.php?option=com_content&task=view&id=34&parentid=47&Itemid=48)  
Acesso em janeiro de 2009.
- <sup>54</sup> A. Lindoso, L. Entrena. "High performance FPGA-based image correlation". *J. Real-Time Image Proc.*, vol. 2, pp. 223–233, 2007.
- <sup>55</sup> Documento Técnico completo com as especificações das FPGAs da família Stratix III  
[http://www.altera.com/literature/hb/stx3/stratix3\\_handbook.pdf](http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf). Acesso em fevereiro de 2008.
- <sup>56</sup> F. de Souza. "Detecção de Falhas em Sistema de Distribuição de Energia Elétrica Usando Dispositivos Programáveis". Programa de Pós-Graduação em Engenharia Elétrica, Unesp, 2008.
- <sup>57</sup> R. Bertrand. *Online Radio & Electronics Course* – Supplementary Reading for 27 Oscillators .  
<http://www.radioelectronicschool.com>, abril, 2002. Acesso em janeiro de 2009.
- <sup>58</sup> Documento técnico de especificações do *software* ModelSim da empresa MentorGraphics.  
[http://www.model.com/products/pdf/datasheets/datasheet\\_modelsim-64.pdf](http://www.model.com/products/pdf/datasheets/datasheet_modelsim-64.pdf). Acesso em março de 2008
- <sup>59</sup> Site da empresa desenvolvedora de softwares para eletrônica, MentorGraphics  
<http://www.mentor.com/>. Acesso em março de 2008.
- <sup>60</sup> S. Carlson, *Introduction to HDL-Based Design Using VHDL*. EUA. Synopsys Inc, 1991.
- <sup>61</sup> D. L. Perry. *VHDL: Programming by Example*. EUA. McGrawHill, 2002
- <sup>62</sup> R. J. TOCCI. *Digital Systems: Principles and Applications*. 1.ed. EUA. Prentice-Hall Inc., 1995.
- <sup>63</sup> K. J. Breeding. *Digital Design Fundamentals*". 2.ed. EUA. Prentice-Hall Inc., 1992.
- <sup>64</sup> J. F. Wakerly. *Digital Design: Principles & Praticice*. 3.ed. New Jersey: Prentice-Hall Inc., 1999.
- <sup>65</sup> D. D. Gajski. *Principles of Digital Design*. 1.ed. New Jersey: Prentice-Hall Inc., 1997.
- <sup>66</sup> R. d'Amore. *VHDL – Descrição e Síntese de Circuitos Digitais*. Brasil, LTC, 1999.
- <sup>67</sup> Manual Técnico do *software* QuartusII [http://www.altera.com/literature/manual/intro\\_to\\_quartus2.pdf](http://www.altera.com/literature/manual/intro_to_quartus2.pdf)  
Acesso em janeiro de 2009.
- <sup>68</sup> Nota de aplicação da Altera especificando o fluxo de projeto integrado ao software de simulação ModelSIM.  
<http://home.eng.iastate.edu/~zzhang/courses/cpre581-f05/resources/modelsim.pdf>.  
Acesso em outubro de 2007.
- <sup>69</sup> Documento técnico da Altera comparando as diferenças entre as versões gratuita e completa do *software* QuartusII. <http://www.altera.com/products/software/quartus-ii/compare-quartus-editions.pdf>  
Acesso em janeiro de 2009.
- <sup>70</sup> V. Bonato. *Proposta de uma arquitetura de hardware em FPGA implementada para SLAM com multi-câmeras aplicada à robótica móvel*. Tese de doutorado, USP, 2008.
- <sup>71</sup> M. Ma, A. van Genderen, P. Beukelman. "A Sign Bit Only Phase Normalization for Rotation and Scale Invariant Template Matching". *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc*, pp. 641-646, 2005.

- 
- <sup>72</sup> Artigo técnico de divulgação da tecnologia 40nm para as FPGAs da família Stratix IV. <http://www.altera.com/literature/wp/wp-01058-stratix-iv-40nm-process-node-custom-logic-devices.pdf> Acesso em janeiro de 2009.
- <sup>73</sup> Informações técnicas sobre performance para as FPGAs da família Stratix IV. <http://www.altera.com/products/devices/stratix-fpgas/stratix-iv/overview/performance/stxiv-performance.html>. Acesso em janeiro de 2009.
- <sup>74</sup> Informações sobre as unidades lógicas das FPGAs Stratix IV. <http://www.altera.com/products/devices/stratix-fpgas/stratix-iv/overview/architecture/stxiv-logic-efficiency.html> Acesso em janeiro de 2009.
- <sup>75</sup> Documento com detalhes sobre o kit de desenvolvimento para FPGAs de alto desempenho da empresa Achronix. [http://www.achronix.com/index.php?option=com\\_content&task=view&id=121&parentid=26&Itemid=67](http://www.achronix.com/index.php?option=com_content&task=view&id=121&parentid=26&Itemid=67) Acesso em janeiro de 2009.
- <sup>76</sup> Y J Ren, J G Zhu, X Y Yang , S H Ye. "The Application of Virtex-II Pro FPGA in High-Speed Image Processing Technology of Robot Vision Sensor". *Journal of Physics: Conference Series*, Volume 48, Issue 1, pp. 373-378, 2006.
- <sup>77</sup> Documento técnico divulgado por PCI Express e PCI-SIG, marcas da PCI-SIG.2005, com especificações do protocolo de comunicação. Revisão 1.1. [www.pcisig.com](http://www.pcisig.com).
- <sup>78</sup> Manual técnico das FPGAs da família StratixIV [http://www.altera.com/literature/hb/stratix-iv/stx4\\_5v1.pdf](http://www.altera.com/literature/hb/stratix-iv/stx4_5v1.pdf) Acesso em janeiro de 2009.
- <sup>79</sup> Documento técnico com informações sobre o kit de desenvolvimento da Altera com ferramenta de comunicação PCI Express <http://www.altera.com/literature/manual/mnl-s2gx-pci-express-devkit.pdf>. Acesso em janeiro de 2009.
- <sup>80</sup> Informações técnicas sobre a família de dispositivos da FPGA de alto desempenho da Xilinx, Virtex-5 <http://www.xilinx.com/products/virtex5/> Acesso em janeiro de 2009.
- <sup>81</sup> Empresa parceira da Xilinx na fabricação de placas acessórios e kits de desenvolvimento <http://www.hitechglobal.com/> Acesso em janeiro de 2009.
- <sup>82</sup> Informações técnicas do kit de desenvolvimento, *PCI Express* com FPGA Virtex-5 da Xilinx <http://www.hitechglobal.com/Boards/PCIExpressLX330T.htm> Acesso em janeiro de 2009.
- <sup>83</sup> Documento técnico *Stratix III Development Kit Reference Manual. Versão1.2*, setembro, 2008. [www.altera.com/literature/manual/rm\\_stratixiii\\_dev\\_kit\\_host\\_board.pdf](http://www.altera.com/literature/manual/rm_stratixiii_dev_kit_host_board.pdf). Acesso em janeiro de 2009.
- <sup>84</sup> Informações técnicas sobre a placa eletrônica fabricada pela empresa Bitec com entrada de vídeo, tipo placa-filha para conexão com kits de desenvolvimento da Altera. [http://www.bitec.ltd.uk/hsmc\\_bvdc.html](http://www.bitec.ltd.uk/hsmc_bvdc.html). Acesso em janeiro de 2009.